NAVSWC MP 91-404

FORTH GRAPHICS TOOLBOX (A USER'S GUIDE FOR USE WITH RFF FORTH)

BY HANSEOK KO

UNDERWATER SYSTEMS DEPARTMENT

14 JUNE 1991

NAVAL SURFACE WARFARE CENTER

Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

NAVSWC MP 91-404

FORTH GRAPHICS TOOLBOX
(A USER'S GUIDE FOR USE WITH RFF FORTH)

BY HANSEOK KO
UNDERWATER SYSTEMS DEPARTMENT

14 JUNE 1991

NAVAL SURFACE WARFARE CENTER
Dahlgren, Virginia 22448-5000 • Silver Spring, Maryland 20903-5000

FOREWORD

The FORTH GRAPHICS TOOLBOX is to be used to develop FORTH based application software. This document is intended to provide a manual detailing the procedures and usage.

This document has been reviewed by the users in the Simulation and Training Section for its technical integrity and the Underwater Signal Processing Branch's line management for elements of format and style.

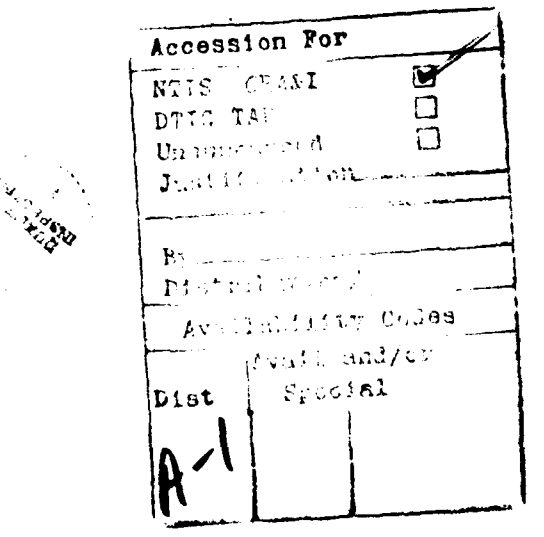
The GRAPHICS TOOLBOX has evolved over a period of two years with input from many users. The author would like to extend his thanks to several people for their input. Kit Yan is credited with the development of many graphics routines at the beginning of this project; the entire project was made much easier because of the strong foundation laid out initially by Yan. Bob Davis, Phil Craun, and Paul Craun provided much useful technical insight in the development of this document. Finally, Ira Rosenbaum, MarkWilliams, Bob Otte, and John Sherman provided the encouragement to the author in pursuing this project.

If you have questions or comments about the TOOLBOX, please contact U25 (Hanseok Ko), (301)394-2372. Comments are welcome and will be considered when the TOOLBOX is revised.

Approved by:

C. Kalivretenos

C. Kalivretenos, Deputy Department Head
Underwater Systems Department



CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1
INTRODUCTION

FORTH GRAPHICS TOOLBOX is a rich collection of graphics routines immediately useful for all FORTH-based application software running on IBM-PC clone microcomputers. The routines are built based on graphics related primitives of both video BIOS call functions and Direct-video functions. The user can develop more exotic application software based on the routines listed in this package.

The central features of the FORTH GRAPHICS TOOLBOX are functions for:

- Video screen environment setup.
- Direct and BIOS called video drawing of a point, a line, and a circle.
- Direct and BIOS called drawing application.
- Demonstration of the package's graphics capabilities.

These functions are implemented in the FORTH environment under the file name GRAPHICS.SEQ. Accessing this file will allow the user to make changes, add features, or learn how a given algorithm works. New application routines can be developed easily by first loading GRAPHICS.SEQ and experimenting with the application SEQ-files.

The GRAPHICS TOOLBOX has evolved over a period of 2 years with input from many users, including those mentioned in the Foreword.

CHAPTER 2
INSTALLATION

FORTH GRAPHICS TOOLBOX can be installed by first getting into RFF FORTH working space. RFF[1] is a 16-bit FORTH system built upon the work of several people: the original F83 system by Laxen and Perry, the "FF" system by Tom Zimmer et. al., and numerous contributions of Robert Davis. FORTH[2] is a language that begins with a powerful set of standard commands, then provides the mechanism by which a user can define his/her own commands. The structured process of building definitions upon previous definitions is the FORTH equivalent of high-level coding. Alternatively, words may be defined directly in assembler mnemonics, using FORTH's assembler. All commands are interpreted by the same interpreter and compiled by the same compiler. The user can get into the RFF FORTH environment by typing 'RFF' from a directory containing the 'RFF.EXE' file.

Once in the RFF FORTH environment, type:

FLOAD GRAPHICS

to load GRAPHICS.SEQ. After GRAPHICS.SEQ is loaded, test it by invoking a demonstration graphics routine such as "TELLIPSE." Upon invoking "TELLIPSE", the user should see random sets of different colored concentric rings displayed on the screen.

CHAPTER 3
HOW TO READ THE TOOLBOX

Each routine or function, presented in Reference A, is tagged as either CODE or WORD. CODE implies that it is an assembly routine that exists because it is either a BIOS called function or an attempt to save processing time. WORD implies that it is colon (:) defined and structured in order to get the full advantage of the FORTH environment. "Category" is listed to provide a quick reference to the routine's background such as whether it is a direct video or BIOS called function.

A stack diagram is provided adjacent to the name of each routine in the parenthesis. For example, the first line for routine LINE looks like this:

LINE	(x1 y1 x2 y2 color # - -)

The order of inputs typed onto the screen is important since it determines the inputs' respective positions on the stack. In the above case, the computer performs the operation in accordance to the task defined by LINE by either pushing or popping the numbers on the stack. The FORTH's stack is described as "last-in, first-out" (LIFO). This means that the only accessible value at any given time is the top value. The system reads input from left to right and executes each word in turn. For input, the rightmost value on the screen will end up on top of the stack. For output, the rightmost value on the screen came from the lowest position on the non-empty column of the stack. The order of inputs with respect to the top of the stack, for the LINE routine, is as follows:

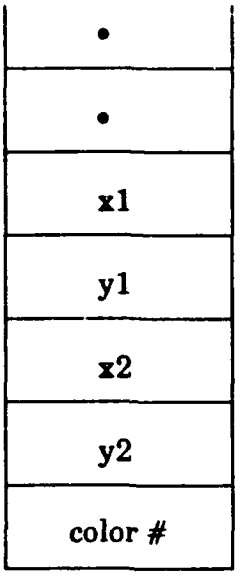


FIGURE 3-1. FORTH STACK DIAGRAM

If a numerical output is desired, the corresponding output variables are listed to the right of the dash (--) in the stack diagram. But if no output variable is listed, as in the case of LINE routine, then an action on the hardware such as "drawing" is expected as the output.

The ranges of the legitimate numerical values are indicated in the Description block. Most values must be given as integers; however, some routines require real numbers as input. When real numbers are required, the input variables in the stack diagram will be denoted by a decimal point as shown below.

AUTOSCALE	(x1. x2. x y color npt hv --)

In general, there are two categories of graphic routines: the direct-video and the BIOS call. The BIOS call routines are denoted by a " _BIOS " postfix attached to the syntax of

the direct-video counterparts. For example, direct-video's AST routine has BIOS call counterpart AST_BIOS which is invoked with software interrupt 10H.

The direct-video routines are at least 10 times faster than the BIOS call routines in getting the corresponding image on the screen. However, the BIOS call routines may become handy if there is a mismatch between the direct-video routines and the display mode type. For example, a program using only BIOS function calls for video output will run in almost any MS-DOS environment, regardless of the video hardware, including (but not limited to) the entire IBM PC and PS/2 family.

The available colors are simple combinations of the primary colors red, green, and blue mapped into 16 colors as follows:

0 = = Black
1 = = Blue
2 = = Green
3 = = Cyan
4 = = Red
5 = = Violet
6 = = Yellow (brown)
7 = = White
8 = = Black (gray)
9 = = Intense blue
10 = = Intense green
11 = = Intense cyan
12 = = Intense red
13 = = Intense violet
14 = = Intense yellow
15 = = Intense white

CHAPTER 4

FORTH GRAPHICS TOOLBOX REFERENCE

This chapter contains a listing of all FORTH Graphics Toolbox routines grouped by subject, listed in alphabetical order, and followed by a brief description of the routine.

TABLE 4-1. VIDEO SCREEN ENVIRONMENT SETUP ROUTINES

VIDEO SCREEN ENVIRONMENT SETUP ROUTINES	
AND_VIDEO	Latched pixels ADed
CGA_HI	640x200 2-color CGA
CO80	Switch to text mode
EGA_HI	640x350 16-color EGA
EGA_LO	640x200 16-color EGA
8x8FONT	Set 8-pixel font
8x14FONT	Set 14-pixel font
8x16FONT	Set 16-pixel font
NORMAL_VIDEO	Latched pixels replaced
OR_VIDEO	Latched pixels ORed
SET_GRAPHMODE	Set the screen to VGA/EGA/CGA etc.
XOR_VIDEO	Latched pixels XORed
VGA_HI	640x480 16-color VGA

TABLE 4-2. DIRECT VIDEO DRAWING COMMANDS

DIRECT VIDEO DRAWING COMMANDS	
AND_VIDEO	Latched pixels ADed
ELLIPSE	Draws an ellipse/circle
HORIZ_LINE	Draws an horizontal staight line
LINE	Draws a stght line of any orientation
VERT_LINE	Draws a vertical straight line
CHRPLOT	Plot a character

TABLE 4-3. BIOS CALLED VIDEO DRAWING COMMANDS

BIOS CALLED VIDEO DRAWING COMMANDS	
ELLIPSE_BIOS	Draws an ellipse/circle
HORIZ_LINE_BIOS	Draws an horizontal staight line
LINE_BIOS	Draws a stght line of any orientation
PUT_PIXEL	Plot a point
VERT_LINE_BIOS	Draws a vertical straight line
CHRPLOT_BIOS	Plot a character

TABLE 4-4. DIRECT VIDEO DRAWING APPLICATION COMMANDS

DIRECT VIDEO DRAWING APPLICATION COMMANDS	
AUTOSCALE	Draws a scale with tick marks
OUTTEXTXY	Plot a string of characters
SIGPLOTB	Plot a string of characters

TABLE 4-5. BIOS CALLED VIDEO DRAWING APPLICATION COMMANDS

BIOS CALLED VIDEO DRAWING APPLICATION COMMANDS	
AUTOSCALE_BIOS	Draws a scale with tick marks
OUTTEXTXY_BIOS	Plot a string of characters

TABLE 4-6. DIRECT VIDEO DEMO ROUTINES

DIRECT VIDEO DEMO ROUTINES	
AMERICA	Writes texts in graphics mode
AST	Draws a set of scales with tick marks
FIREWORK1	Draws random flashes of fireballs
FIREWORK3	Draws random flashes of fireballs
TELLIPSE	Draws random sets of rings
TLINE	Draws random sets of lines
TXT	Writes texts in four orientations

TABLE 4-7. BIOS CALLED VIDEO DEMO ROUTINES

BIOS CALLED VIDEO DEMO ROUTINES	
AMERICA_BIOS	Writes texts in graphics mode
AST_BIOS	Draws a set of scales with tick marks
FIREWORK2	Draws random flashes of fireballs
TLINE_BIOS	Draws random sets of lines
TXT_BIOS	Writes texts in four orientations

TABLE 4-8. BIOS FUNCTION CALLS

BIOS FUNCTION CALLS	
FONTAD	Get FONT address within EGA/VGA
GET_GRAPHMODE	Get info about current graphic mode
GET_PIXEL	Read color info at (x,y) pixel location
GET_XY	Read current cursor position
GOTO_XY	Move cursor to (x,y) position
PUT_PIXEL	Plot a point at (x,y) pixel location
READ_CHAR	Read character at current cursor position
SCROLL_PAGEDOWN	Scroll texts top to bottom on screen
SCROLL_PAGEUP	Scroll texts bottom to top on screen
SET_ACTIVEPAGE	Select a page as active page for graphics
SET_BORDER	Draw a border line
SET_GRAPHMODE	Set the screen to various graphics modes
SET_PALETT	Change palette's color entry to new color
WRITE_CHAR	Put character at current cursor position
WRITE_TCHAR	Put character at current cursor position and move cursor to next position

REFERENCES

1. Davis, Robert H., "RFF - A 16-Bit Memory Model for Large Address Space Computers," Proceedings of SIGFORTH Applications Symposium, Austin, TX, Feb 1989.

2. Brodie, Leo, Starting FORTH, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

APPENDIX A

FORTH GRAPHICS TOOLBOX
FUNCTIONAL DESCRIPTION

AMERICA (x y --)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:

Demonstration of graphics text capability.

Description:

After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.)
is invoked and given (x,y) coordinates, AMERICA
writes text "UNITED STATES OF AMERICA" in four
90-deg orientations centered at (X,Y).

Examples:

VGA_HI 300 200 AMERICA

See also:

AMERICA_BIOS

AMERICA_BIOS (x y --)

Type: WORD

Category: BIOS call video graphic drawing routine

Purpose:

Demonstration of graphics text capability.

Description:

After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.)
is invoked and given (x,y) coordinates, AMERICA_BIOS
writes text "UNITED STATES OF AMERICA" in four
90-deg orientations centered at (X,Y).

Examples:

VGA_HI 300 200 AMERICA_BIOS

See also:

AMERICA

AND_VIDEO (--)

Type: WORD

Category: Video environment controlling routine

Purpose:

Specify functions (AND, OR, XOR) available for updating pixels during pixel write modes.

Description:

AND_VIDEO specifies the Data Rotate/Function Select register's (03H) 2-bit fields to 00001000. This bit pattern forces latched pixels to be ANDed when updated. Note that the variable PIXEL_MODE stores the 2-bit fields.

Examples:

```
VGA_HI AND_VIDEO 250 300 12 PUT_PIXEGA
```

See also:

OR_VIDEO, XOR_VIDEO

AST (x. y. --)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:

Demonstration of graphics drawing capability.

Description:

After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.) is invoked and given the two boundaries (x1.,x2.) with decimal numbers, AST will draw a set of differently scaled axes.

Examples:

```
VGA_HI  20.0 2000.0 AST
```

See also:

AST_BIOS

AST_BIOS (x. y. --)

Type: WORD

Category: BIOS call graphic drawing routine

Purpose:

Demonstration of graphics drawing capability.

Description:

After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.)
is invoked and given the two boundaries (x1.,x2.) with decimal
numbers, AST_BIOS will draw a set of differently scaled axes.

Examples:

```
VGA_HI  20.0 2000.0 AST_BIOS
```

See also:

AST

AUTOSCALE (x1. x2. x y color npt hv --)

Type: WORD

Category: Direct graphics routine

Purpose:

Draw scales with tick marks

Description:

Given two real numbers x1. and x2., AUTOSCALE draws a
straight line between the two numbers with tick marks and
labels whose sizes are automatically adjusted to minimize
cluttering.

AUTOSCALE requires the following inputs:

x1. == lower limit of two real numbers

x2. == upper limit of two real numbers

x == x-coordinate of the line's starting point

y == y-coordinate of the line's starting point

color == color number (0, 15)

npt == length of the line in terms of the number of pixels
(i.e., (1,640) for horizontal orientation and (1,480)
for vertical orientation)

hv == 1 for horizontal
0 for vertical

Examples:

```
0. 1000.0 20 250 12 300 1 AUTOSCALE
```

See also:

AUTO_RANGE, AUTOSCALE_BIOS

AUTOSCALE_BIOS (x1. x2. x y color npt hv --)

Type: WORD

Category: Video BIOS call graphics routine

Purpose:

Draw a scale with tick marks

Description:

Given two real numbers, x1. and x2., AUTOSCALE_BIOS draws a straight line between the two numbers with tick marks and labels whose sizes are automatically adjusted to minimize cluttering.

AUTOSCALE_BIOS requires the following inputs:

x1.== lower limit of two real numbers

x2.== upper limit of two real numbers

x == x-coordinate of the line's starting point

y == y-coordinate of the line's starting point

color == color number (0, 15)

npt == length of the line in the number of pixels (i.e., (1,640) for horizontal orientation and (1,480) for vertical orientation)

hv == 1 for horizontal
0 for vertical

Examples:

0. 1000.0 20 250 12 300 1 AUTOSCALE_BIOS

See also:

AUTO_RANGE, AUTOSCALE

CGA_HI (--)

Type: WORD

Category: Video screen environment setup routine

Purpose:

Set the system to the specified graphics mode

Description:

Invoking CGA_HI will bring the screen to high-resolution-CGA mode by calling SET_GRAPHMODE, a video BIOS function routine. This call corresponds to setting the video to:

640x200 2-color graphics: one must be black.

Examples:

CGA_HI 50 50 200 200 12 LINE_BIOS

See also:

CGA_HI, EGA_HI, EGA_LO, VGA_HI, VGA_LO

CHRPLOT (charn color# x y liml limh hv --)

Type: WORD

Category: Direct video character drawing routine

Purpose:

Allow characters to be drawn in graphics mode on graphics coordinates.

Description:

Upon invoking CHRPLOT during graphics mode, a character will be drawn in pixel coordinates (x,y) in either horizontal or vertical orientation.

Input requirement:

```
charn == IBM PC character set coded in decimal
         A=65
         B=66
         C=67
         .
         .
         Z=90
         a=97
         b=98
         .
         z=122

color == color# (0,15)
         (See "How to read the toolbox" for color codes)
   x == character position's x-coordinate [(0,639) for VGA_HI]

   y == character position's y-coordinate [(0,479) for VGA_HI]
liml == bottom window clip == 0
limh == top window clip  (i.e., 639 for horizontal orientation
        and 479 for vertical orientation)

  hv == 0  vertical
        1  horizontal
```

Examples:

```
VGA_HI 8x14FONT 65 14 10 20 0 639 1 CHRPLOT
```

See also:

CHRPLOT_BIOS

CHRPLOT_BIOS (charn color# x y liml limh hv --)

Type: WORD

Category: Video BIOS call character drawing routine

Purpose:

Allow characters to be drawn in graphics mode on graphics coordinates.

Description:

Upon invoking CHRPLOT_BIOS during graphics mode, a character will be drawn in pixel coordinates (x,y) in either horizontal or vertical orientation.

Input requirement:

```
charn == IBM PC character set coded in decimal
         A=65
         B=66
         C=67
         .
         .
         Z=90
         a=97
         b=98
         .
         z=122

color == color# (0,15)
         (See "How to read the toolbox" for color codes)
 x    == character position's x-coordinate [(0,639) for VGA_HI]
 y    == character position's y-coordinate [(0,479) for VGA_HI]
 liml == bottom window clip == 0
 limh == top window clip  (i.e., 639 for horizontal orientation
         and 479 for vertical orientation)

 hv   == 0  vertical
         1  horizontal
```

Examples:

```
VGA_HI 65 14 10 20 0 639 1 CHRPLOT_BIOS
```

See also:

CHRPLOT

CO80 (--)

Type: WORD

Category: Video screen environment setup routine

Purpose:

Switch from screen's graphic mode to text mode.

Description:

Invoking CO80 will bring the screen to 80 column alphanumeric (e.g., text) mode by calling SET_GRAPHMODE, a video BIOS function routine.

During the process, the screen gets cleared.

Examples:

```
CO80
```

See also:

CGA_HI, EGA_HI, EGA_LO, VGA_HI, VGA_LO

EGA_HI (--)

Type: WORD

Category: Video screen environment setup routine

Purpose:

Set the system to the specified graphics mode

Description:

Invoking EGA_HI will bring the screen to high-resolution-EGA mode by calling SET_GRAPHMODE, a video BIOS function routine. This call corresponds to setting the video to:

640x350 16-color graphics

It also sets up the graphics character heights to be 14 units (=pixels) high. (VGA_HI has 16 units high character size)

During the process, the screen gets cleared.

Examples:

```
EGA_HI   50 50 200 200 12 LINE_BIOS
```

See also:

CGA_HI, CGA_LO, EGA_LO, VGA_HI, VGA_LO

NAVSWC MP 91-404

EGA_LO (--)

Type: WORD

Category: Video screen environment setup routine

Purpose:

Set the system to the specified graphics mode

Description:

Invoking EGA_LO will bring the screen to low-resolution-EGA mode by calling SET_GRAPHMODE, a video BIOS function routine. This call corresponds to setting the video to:

640x200 16-color graphics

It also sets up the graphics character heights to be 14 units high. (VGA_HI has 16 units high character size)

During the process, the screen gets cleared.

Examples.

```
EGA_LO   50 50 200 200 12 LINE_BIOS
```

See also:

CGA_HI, EGA_LO, VGA_HI, VGA_LO

EGAPIX_ADDR (x y --)

Type: LABEL

Category: Assembly code subroutine needed for direct-video graphics routines

Purpose: Determine buffer address of pixel in native EGA/VGA modes:
320x200 16-color 640x200 16-color 640x350 16-color
640x350 monochrome (4-color) 640x480 2-color 640x480 16-color

Description:
This subroutine converts pixel coordinates to the corresponding byte and bit offsets in the video buffer. In graphics modes, the video buffer can be thought of as a flat, two-dimensional array of pixels with its origin at the upper left corner. What is visible on the screen is a subset of the pixels represented in the buffer. On the CGA, the video buffer can contain only one screenful of pixels, so the first byte in the buffer represents the pixels in the screen's upper left corner. On the EGA, MCGA, and VGA, however, the video buffer can store several screenfuls of pixels. You can thus select which portion of the video buffer appears on the screen.

Every pixel on the screen can be identified by a unique pair of (x,y) coordinates relative to the screen's upper left corner. Each (x,y) pair also corresponds to a particular byte offset in the video buffer and a bit offset in that byte. Thus, given a pixel's (x,y) coordinates on the screen, you can compute where in the video buffer the pixel is represented. Transforming pixel coordinates to a buffer offset involves simple logic. Begin by calculating the offset of the start of pixel row y. (For CGA and Hercules graphics modes, this calculation accounts for the interleaving of the video buffer.) To this value, add the byte offset of the xth pixel in the row. Finally, add the byte offset of the start of the displayed portion of the video buffer to obtain the final byte offset of the pixel.
PixelByteOffset = RowOffset(y) + ByteOffset(x) + OriginOffset
The bit offset of the pixel within the byte that contains its value depends only on the number of pixels represented in each byte of the video buffer. However, it is more practical to represent a pixel's bit offset as a bit mask rather than as an ordinal bit number. This is done easily with a logical shift instruction.

Examples:

```
---
CALL EGAPIX_ADDR
---
```

See also:
PUT_PIXEGA, GET_PIXEGA, VERT_LINE, HORIZ_LINE, LS_LINE, etc for examples of usage.

ELLIPSE (color# b a yc xc --)

Type: CODE

Category: Direct-video drawing routine

Purpose:

Provide an ellipse/circle drawing capability

Description:

This routine draws an ellipse in EGA/VGA graphics modes with the following inputs:

color# == (0,15) (SEE "How to read the toolbox" for color codes)
b == minor axis,
a == major axis,
xc == x-coordinate of ellipse's center
yc == y-coordinate of ellipse's center

Examples:

12 50 100 300 300 ELLIPSE to draw an ellipse

12 50 50 300 300 ELLIPSE to draw a circle

See also:

LINE, LINE_BIOS

FIREWORK1 (--)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:

Demonstration of graphics drawing capability.

Description:

After VGA_HI is invoked, FIREWORK1 will draw random flashes of fireballs resembling fireworks lighting up in the sky. The fireball's textures are made up of rays of different color.

Examples:

VGA_HI FIREWORK1

See also:

FIREWORK2, FIREWORK3

NAVSWC MP 91-404

FIREWORK2 (--)

Type: WORD

Category: BIOS call video graphic drawing routine

Purpose:

Demonstration of graphics drawing capability.

Description:

After VGA_HI is invoked, FIREWORK2 will draw random flashes of fireballs resembling fireworks lighting up the sky. The fireball's textures are made up of rays of same color.

Examples:

VGA_HI FIREWORK2

See also:

FIREWORK1, FIREWORK3

FIREWORK3 (--)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:

Demonstration of graphics drawing capability.

Description:

After VGA_HI is invoked, FIREWORK3 will draw random flashes of fireballs resembling fireworks lighting up the sky. The fireball's textures are made up of rays of same color.

Examples:

VGA_HI FIREWORK3

See also:

FIREWORK1, FIREWORK2

8x8FONT (--)

Type: WORD

Category: General

Purpose:

Allow the use of different size fonts when using
CHRPLOT (character plotting routine)

Description:

Invoking 8x8FONT returns the address of where the
font character is and the height of each character.

Input : none

Output: FONZ == the address where the font is located
CHRHGHT == the height (8 pixels) of the font in pixels

Examples:

VGA_HI 8x8FONT 65 14 200 200 0 639 1 CHRPLOT

See also:

8x14FONT, 8x16FONT

8x14FONT (--)

Type: WORD

Category: General

Purpose:

Allow the use of different size fonts when using
CHRPLOT (character plotting routine)

Description:

Invoking 8x14FONT returns the address of where the
font character is and the height of each character.

Input : none

Output: FONZ == the address where the font is located
CHRHGHT == the height (14 pixels) of the font in pixels

Examples:

VGA_HI 8x14FONT 65 14 200 200 0 639 1 CHRPLOT

See also:

8x8FONT, 8x16FONT

8x16FONT (--)

Type: WORD

Category: General

Purpose:

Allow the use of different size fonts when using CHRPLOT (character plotting routine)

Description:

Invoking 8x16FONT returns the address of where the font character is and the height of each character.

Input : none

Output: FONT2 == the address where the font is located
CHRHGHT == the height (16 pixels) of the font in pixels

Examples:

VGA_HI 8x16FONT 65 14 200 200 0 639 1 CHRPLOT

See also:

8x8FONT, 8x14FONT

FONTAD (-- seg offset)

Type: CODE

Category: BIOS function call 17
Subservice call 30

Purpose:

Get address of FONT located within the EGA/VGA BIOS

Description:

FONTAD is a BIOS function call #11H routine which allows to get the following current character generator information (i.e AL=30H).

Possible input:
BH = 2 Address of 8x14 character table
= 3 Address of 8x8 character table
= 4 Address of second half of 8x8 character table
= 5 Address of 9x14 alternate character table
= 6 Address of 8x16 character table
= 7 Address of 9x16 alternate character table

Returned values:

ES:BP = address of character definition table

Examples:

0200 FONTAD
(returns location of FONT in SEGMENT and OFFSET as used by 8x16FONT or 8x14FONT)

See also:

8x16FONT, 8x14FONT, 8x8FONT

GET_GRAPHMODE (-- video mode)

Type: CODE

Category: BIOS function call 15

Purpose:

Get information about the current graphic mode

Description:

GET_GRAPHMODE is a BIOS function call routine which imports the current graph mode information.

Input : none

Possible Output:

```
0 & 1 - 40 column alphanumeric (CGA compatible)
2 & 3 - 80 column alphanumeric (CGA compatible)
4 & 5 - 320x200 4-color graphics limited to 2 palettes (CGALo)
    6 - 640x200 2-color graphics: one must be black (CGAHi)
    7 - monochrome alphanumeric (monochrom adapter compatible)
    8 - 12 - reserved
   13 - 320x200 16 color
   14 - 640x200 16 color (EGALo)
   15 - 640x350 monochrome graphics (EGAMonoHi)
   16 - 640x350 16 color (EGAHi)
   17 - 640x480 monochrome graphics (VGAMonoHi)
   18 - 640x480 16 color (VGAHi)
   19 - 320x200 256 color (VGA only)
```

Examples:

```
GET_GRAPHMODE
```

See also:

SET_GRAPHMODE

GET_PIXEGA (x y -- color)

Type: CODE

Category: Direct video

Purpose:

Get color information at the indicated pixel location

Description:

GET_PIXEGA is a direct video routine which imports the color# (0,15) at (x,y) coordinate pixel position.

Input :

x == x-coordinate (0,639)

y == y-coordinate (0,479)

Output: color code numbers (0,15)
(See "How to read the toolbox" for color codes)

Examples:

```
VGA_HI TELLIPSE     (draws many sets of ellipse rings)
40 50 GET_PIXEGA    (color# for pixel(40,50) is retrieved)
```

See also:
GET_PIXEL (equivalent BIOS call routine)
PUT_PIXEGA

GET_PIXEL (x y -- color#)

Type: CODE

Category: Video BIOS function call 13

Purpose:

Read color information at the indicated pixel location

Description:

GET_PIXEL is a video BIOS call routine which imports the color# (0,15) at (x,y) coordinate pixel position.

Input :

x == x-coordinate (0,639)

y == y-coordinate (0,479)

Output: color code numbers (0,15)
(See "How to read the toolbox" for color codes)

Examples:

```
VGA_HI TELLIPSE     (draws many sets of ellipse rings)
40 50 GET_PIXEL     (color# for pixel(40,50) is retrieved)
```

See also:
GET_PIXEGA (equivalent direct-video routine)
PUT_PIXEGA

GET_XY (-- x y)

Type: CODE

Category: Video BIOS function call 3

Purpose:

Read current cursor position

Description:

GET_XY is a video BIOS call routine which imports the (x,y) position coordinates of the cursor.

Input : none

Output:

x == x-coordinate (0,79)

y == y-coordinate (0,24)

Examples:

```
GET_XY
```

See also:

GOTO_XY

GOTO_XY (x y --)

Type: CODE

Category: Video BIOS function call 2

Purpose:

Move cursor position to the designated coordinates (x,y)

Description:

GOTO_XY is a video BIOS call routine which allows positioning the cursor to the coordinates (x,y).

Input:

x == x-coordinate (0,79)

y == y-coordinate (0,24)

Output: none

Examples:

GOTO_XY

See also:

GET_XY

HORIZ_LINE (y x1 x2 color# --)

Type: CODE

Category: Direct-video drawing routine

Purpose:

Provide a horizontal (slope=0) straight line drawing capability

Description:

This routine draws an horizontal line in EGA/VGA graphics modes with the following inputs:

```
    y  = vertical position  [i.e., (0,480) if VGA_HI]
   x1  = start point in the x-axis ((0,640) if VGA_HI]
   x2  = end point in the x-axis  ((0,640) if VGA_HI]
color# = line's color (1,15)
         (See "How to read the toolbox" for color codes)
```

Examples:

VGA_HI 100 20 300 12 HORIZ_LINE

See also:

HORIZ_LINE_BIOS, VERT_LINE, LINE,

HORIZ_LINE_BIOS (y x1 x2 color# --)

Type: WORD

Category: Video BIOS call drawing routine

Purpose:

Provide a horizontal (i.e, slope=0) straight line drawing capability

Description:

This routine draws an horizontal line in EGA/VGA graphics modes with the following inputs:

```
  y  = vertical position [i.e., (0,480) if VGA_HI]
  x1 = start point in the x-axis [(0,640) if VGA_HI]
  x2 = end point in the x-axis [(0,640) if VGA_HI]
color# = line's color (1,15)
       (See "How to read the toolbox" for color codes)
```

Examples:

```
VGA_HI 100 20 300 12 HORIZ_LINE_BIOS
```

See also:

HORIZ_LINE, VERT_LINE, LINE, LINE_BIOS

HS_LINE (x1 y1 x2 y2 color# --)

Type: CODE

Category: Direct-video drawing routine

Purpose:

Provide a straight line (slope >=1) drawing capability and supports LINE routine

Description:

This routine draws a sloped line in EGA/VGA graphics modes with the following inputs:

```
 x1  = start point in the x-axis [(0,640) if VGA_HI]
 y1  = start point in the y-axis [i.e., (0,480) if VGA_HI]
 x2  = end point in the x-axis [ (0,640) if VGA_HI]
 y2  = end point in the y-axis [i.e., (0,480) if VGA_HI]
color# = line's color (1,15)
       (See "How to read the toolbox" for color codes)
```

Examples:

```
VGA_HI 100 20 300 12 HORIZ_LINE
```

See also:

LINE, LS_LINE, HORIZ_LINE_BIOS, VERT_LINE, LINE,

LINE (x1 y1 x2 y2 color# --)

Type: CODE

Category: Direct-video drawing routine

Purpose:

Provide the capability of drawing a straight line in any orientation.

Description:

This routine draws a straight line in EGA/VGA graphics modes with the following inputs:

x1 = start point in the x-coordinate [(0,640) if VGA_HI]
y1 = start point in the y-coordinate [(0,480) if VGA_HI]
x2 = end point in the x-coordinate [(0,640) if VGA_HI]
y2 = end point in the y-coordinate [(0,480) if VGA_HI]
color# = line's color (1,15)
(See "How to read the toolbox" for color codes)

Examples:

VGA_HI 20 20 300 300 12 HORIZ_LINE

See also:

LS_LINE
HS_LINE
HORIZ_LINE,
HORIZ_LINE_BIOS,
VERT_LINE,
VERT_LINE_BIOS,
LINE_BIOS,
ELLIPSE

LINE_BIOS (x1 y1 x2 y2 color# --)

Type: WORD

Category: Video BIOS call drawing routine

Purpose:

Provide the capability of drawing a straight line in any orientation.

Description:

This routine draws a straight line in EGA/VGA graphics modes with the following inputs:

x1 = start point in the x-coordinate [(0,640) if VGA_HI]
y1 = start point in the y-coordinate [(0,480) if VGA_HI]
x2 = end point in the x-coordinate [(0,640) if VGA_HI]
y2 = end point in the y-coordinate [(0,480) if VGA_HI]
color# = line's color (1,15)
(See "How to read the toolbox" for color codes)

Examples:

VGA_HI 20 20 300 300 12 HORIZ_LINE

See also:

HORIZ_LINE,
HORIZ_LINE_BIOS,
VERT_LINE,
VERT_LINE_BIOS,
LINE_BIOS,
ELLIPSE

LS_LINE (x1 y1 x2 y2 color# --)

Type: CODE

Category: Direct-video drawing routine

Purpose:

Provide a straight line (slope <=1) drawing capability and supports the LINE routine

Description:

This routine draws a sloped line in EGA/VGA graphics modes with the following inputs:

x1 = start point in the x-axis [(0,640) if VGA_HI]
y1 = start point in the y-axis [i.e., (0,480) if VGA_HI]
x2 = end point in the x-axis [(0,640) if VGA_HI]
y2 = end point in the y-axis [i.e., (0,480) if VGA_HI]
color# = line's color (1,15)
(See "How to read the toolbox" for color codes)

Examples:

VGA_HI 100 20 300 12 HORIZ_LINE

See also:

HS_LINE, HORIZ_LINE_BIOS, VERT_LINE, LINE,

NORMAL_VIDEO (--)

NORMAL_VIDEO (--)

Type: WORD

Category: Video environment controlling routine

Purpose:

Specify functions (AND, OR, XOR) available for updating pixels during pixel write modes.

Description:

NORMAL_VIDEO specifies the Data Rotate/Function Select register's (03H) two bit fields to 00000000. This bit pattern forces latched pixels to be normalized when updated. Note that the variable PIXEL_MODE stores the two bit fields.

Examples:

VGA_HI OR_VIDEO 250 300 12 PUT_PIXEGA NORMAL_VIDEO

See also:

AND_VIDEO, XOR_VIDEO

OR_VIDEO (--)

Type: WORD

Category: Video environment controlling routine

Purpose:

Specify functions (AND, OR, XOR) available for updating pixels during pixel write modes.

Description:

OR_VIDEO specifies the Data Rotate/Function Select register's (03H) two bit fields to 00010000. This bit pattern forces latched pixels to be OR'd when updated. Note that the variable PIXEL_MODE stores the two bit fields.

Examples:

```
VGA_HI OR_VIDEO 250 300 12 PUT_PIXEGA
```

See also:

AND_VIDEO, XOR_VIDEO, NORMAL_VIDEO

OUTTEXTXY (PTR # CNT COLOR XY LIML LIMH FG --)

Type: WORD

Category: Direct-video graphic text drawing routine

Purpose: Write graphics text.

Description:

After VGA_HI is invoked, OUTTEXTXY will write texts in the quotes on the graphics screen with the following designation:

Input:

```
PTR # COUNT == put the texts in this form -> $" abc..text "

      color == color# (0,15)
               (SEE "How to read the toolbox" for color codes)
          x == character position's x-coordinate
               [(0,639) for VGA_HI]
          y == character position's y-coordinate
               [(0,479) for VGA_HI]
       liml == bottom window clip == 0
       limh == top window clip  (i.e., 639 for horizontal
               orientation and 479 for vertical orientation)

         hv == 0  vertical (top to bottom)
               1  horizontal (left to right)
               2  vertical (bottom to top)
               3  horizontal (right to left)
```

Examples:

```
VGA_HI
$" VERTICAL 1" 11 300 200 0 639 0 OUTTEXTXY
$" VERTICAL 2" 12 300 200 0 639 2 OUTTEXTXY
$" HORIZONTAL 1" 13 300 200 0 349 1 OUTTEXTXY
$" HORIZONTAL 2" 14 300 200 0 349 3 OUTTEXTXY
```

See also:

TXT, TXT_BIOS, OUTTEXTXY_BIOS

NAVSWC MP 91-404

OUTTEXTXY_BIOS (PTR # CNT COLOR XY LIML LIMH FG --)

Type: WORD

Category: Video BIOS call graphic text drawing routine

Purpose: Write graphics text.

Description:

After VGA_HI is invoked, OUTTEXTXY_BIOS will write text inside the quotation marks (" ") on the graphics screen with the following designation:

Input:

PTR # COUNT == put text in this form -> $" abc..text "

color == color# (0,15)
(See "How to read the toolbox" for color codes)
x == character position's x-coordinate
[(0,639) for VGA_HI]
y == character position's y-coordinate
[(0,479) for VGA_HI]
liml == bottom window clip == 0
limh == top window clip (i.e., 639 for horizontal orientation and 479 for vertical orientation)

hv == 0 vertical (top to bottom)
1 horizontal (left to right)
2 vertical (bottom to top)
3 horizontal (right to left)

Examples:

```
VGA_HI
$" VERTICAL 1" 11 300 200 0 639 0 OUTTEXTXY_BIOS
$" VERTICAL 2" 12 300 200 0 639 2 OUTTEXTXY_BIOS
$" HORIZONTAL 1" 13 300 200 0 349 1 OUTTEXTXY_BIOS
$" HORIZONTAL 2" 14 300 200 0 349 3 OUTTEXTXY_BIOS
```

See also:

TXT, TXT_BIOS, OUTTEXTXY

PUT_PIXEGA (x y color# --)

Type: CODE

Category: Direct video pixel drawing assembly routine

Purpose:

Plot a point at the indicated pixel location

Description:

PUT_PIXEGA is a video BIOS call routine which plots a point with the color# (0,15) at the specified (x,y) coordinate pixel position.

Input :

x == x-coordinate (0,639)

y == y-coordinate (0,479)

color# == color code numbers (0,15)
(See "How to read the toolbox" for color codes)

Examples:

```
VGA_HI  40 50  12 PUT_PIXEL
```

See also:

GET_PIXEGA
PUT_PIXEL (equivalent BIOS call routine)

PUT_PIXEL (x y color# --)

Type: CODE

Category: Video BIOS function call 12

Purpose:

Plot a point at indicated pixel location

Description:

PUT_PIXEL is a video BIOS call routine which plots a point with the color# (0,15) at the specified (x,y) coordinate pixel position.

Input :

x == x-coordinate (0,639)

y == y-coordinate (0,479)

color# == color code numbers (0,15)
(See "How to read the toolbox" for color codes)

Examples:

VGA_HI 40 50 12 PUT_PIXEL

See also:

GET_PIXEGA
PUT_PIXEGA (equivalent direct-video routine)

RANDOMVECT (maxrk k -- r1 .. rk)

Type: WORD

Category: Random number generator routine

Purpose:

Generate uniform density random numbers

Description:

Given the maximum value and the count, RANDOMVECT will generate integer random numbers in the ranges of [0,maximum value].

Input:

maxrk == maximum value in the set of random numbers

k == count of random numbers

Output:

r1 .. rk == random numbers

Examples:

50 5 RANDOMVECT will result in 3 0 45 34 2

See also:

READ_CHAR (-- char)

Type: CODE

Category: Video BIOS function call 8

Purpose:

Read character at current cursor position

Description:

READ_CHAR is a video BIOS call routine which retrieves the character at the current cursor position.

Input : none

Output:

character (a, b, ...z, A, B, ...)

Examples:

See also:

SCROLL_PAGEDOWN (#oflines --)

Type: CODE

Category: Video BIOS function call 7

Purpose:

Scroll the text on the screen

Description:

This function scrolls the text on the screen - lines move from the top of the screen toward the bottom, and blank lines are inserted at the top.

Input :

# of lines to scroll

Output:

scrolled text lines

Examples:

10 SCROLL_PAGEDOWN

See also:

SCROLL_PAGEUP

SCROLL_PAGEUP (#oflines --)

Type: CODE

Category: Video BIOS function call 6

Purpose:

Scroll the text on the screen

Description:

This function scrolls the text on the screen - lines move from the bottom of the screen toward the top, and blank lines are inserted at the bottom.

Input :

# of lines to scroll

Output:

scrolled text lines

Examples:

10 SCROLL_PAGEUP

See also:

SCROLL_PAGEDOWN

SET_ACTIVEPAGE (page# --)

Type: CODE

Category: Video BIOS function call 5

Purpose:

Selects Page as the active page for graphics output

Description:

This function selects Page as the active page for graphics output. Although the adapter may have several pages (or screens) of information in memory, only one page, the active display page, is visible at any one time. Most of the functions which allow screen modification (write characters, plot points, move the cursor, etc.) also allow selecting which page to modify, thus an invisible screen (non-active display page) may be changed. For example, while displaying one page, another may be created or changed. This feature allows switching to the new screen immediately (a technique useful for animation or slide shows). This function allows choosing which screen is displayed. Usually, screen 0 is the only screen displayed and modified.

Input :

page#

Output:

--

Examples:

3 SET_ACTIVEPAGE

See also:

SET_BORDER (color# --)

Type: CODE

Category: Video BIOS function call 16

Purpose:

Draw a border line with the new color selected

Description:

This function draws a border line with the color selectd.

Input :

color# (0,15)

(See "How to read the toolbox" for color codes)

Output:

--

Examples:

12 SET_BORDER

See also:

SET_PALETTE

SET_GRAPHMODE (graphmode# --)

Type: CODE

Category: Video BIOS function call 0

Purpose:

Set the screen to various graphics mode

Description:

Sets the system to the specified graphics mode as shown below:

0 & 1 - 40 column alphanumeric (CGA compatible)
2 & 3 - 80 column alphanumeric (CGA compatible)
4 & 5 - 320x200 4-color graphics limited to 2 palettes (CGALo)
6 - 640x200 2-color graphics: one must be black (CGAHi)
7 - monochrome alphanumeric (monochrome adapter compatible)
8- 12 - reserved
13 - 320x200 16 color
14 - 640x200 16 color (EGALo)
15 - 640x350 monochrome graphics (EGAMonoHi)
16 - 640x350 16 color (EGAHi)
17 - 640x480 monochrome graphics (VGAMonoHi)
18 - 640x480 16 color (VGAHi)
19 - 320x200 256 color (VGA only)

Input :

color# (0,15)

(See "How to read the toolbox" for color codes)

Output:

--
In effect, invoking this code clears the screen.

Examples:

18 SET_GRAPHMODE

See also:

8x16FONT

SET_PALETTE (palette# color# --)

Type: CODE

Category: Video BIOS function call 16

Purpose:

Change the color number entry setting in the palette to new color

Description:

This function updates a specified palette register to new color.

Input :

palette# == (0,15)
color# == (0,15)

(See "How to read the toolbox" for color codes)

Output:

--

Examples:

12 13 SET_PALETTE

See also:

SET_BORDER

SET_VPLANE (Hex# --)

Type: CODE

Category: Routine

Purpose:

Enable/disable video plane

Description:

SET_VPLANE allows only certain planes within the graphics memory to be drawn or changed. Works only when video mode is set to 0.

Possible Input:

0 == plane # 0

1 == plane # 1

2 == plane # 2

3 == plane # 3

Examples:

3 SET_VPLANE

SIGPLOTB (ptr # cnt color x y lim fg --)

Type: WORD

Category: Direct-video graphic text drawing routine

Purpose: Write graphics text.

Description:

After VGA_HI is invoked, SIGPLOTB will write text inside quotation marks (" ") on the graphics screen with the following designation:

Input:

PTR # COUNT == put the text in this form -> $" abc..text "

color == color# (0,15)
(See "How to read the toolbox" for color codes)
x == character position's x-coordinate
[(0,79) for VGA_HI]
y == y-coordinate pixel position
[(0,479) for VGA_HI]
lim == top window clip (i.e, 0)

FG == 0 == 00 = vertical (top to bottom), or

1 == 01 = horizontal (left to right), or

2 == 10 = vertical (top to bottom), store

3 == 11 = horizontal (left to right), store

Examples:

```
VGA_HI
$" VERTICAL 1" 11 40 200 0  0 SGPLOTB
```

See also:

OUTTEXTXY

TELLIPSE (--)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:

Demonstration of graphics drawing capability.

Description:

After VGA_HI is invoked, TELLIPSE will draw random sets of different colored rings. This is a demo for the ELLIPSE drawing routine.

Examples:

```
VGA_HI  TELLIPSE
```

See also:

FIREWORK2, FIREWORK3

TLINE (x y --)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:

Demonstration of graphics drawing capability.

Description:

After VGA_HI is invoked, TLINE will draw random sets of straight lines in different colors cornered by (x,y). This is a demo for the line drawing routine.

Examples:

```
VGA_HI  100 100 TLINE
```

See also:

FIREWORK2, FIREWORK3, TELLIPSE, TLINE_BIOS

TLINE_BIOS (x y --)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:

Demonstration of graphics drawing capability.

Description:

After VGA_HI is invoked, TLINE_BIOS will draw random sets of straight lines in different colors cornered by (x,y). This is a demo for the line drawing routine.

Examples:

```
VGA_HI  100 100 TLINE_BIOS
```

See also:

FIREWORK2, FIREWORK3, TELLIPSE, TLINE

TXT (--)

Type: WORD

Category: Direct-video graphic drawing routine

Purpose:

Demonstration of graphics text capability.

Description:

After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.), is invoked and given (x,y) coordinates, TXT writes texts " xxxx... " in four 90-deg orientations centered at (300, 200).

Examples:

VGA_HI TXT

See also:

AMERICA, AMERICA_BIOS, TXT_BIOS

TXT_BIOS (--)

Type: WORD

Category: Video BIOS call graphic drawing routine

Purpose:

Demonstration of graphics text capability.

Description:

After GRAPHICS MODE (VGA_HI, EGA_HI, CGA_LO, etc.), is invoked and given (x,y) coordinates, TXT_BIOS writes text " xxxx... " in four 90-deg orientations centered at (300, 200).

Examples:

VGA_HI TXT_BIOS

See also:

AMERICA, AMERICA_BIOS, TXT

VERT_LINE (x y1 y2 color --)

Type: CODE

Category: Direct-video drawing routine

Purpose:

Provide a vertical (slope=infinite) straight line drawing capability

Description:

This routine draws a vertical line in EGA/VGA graphics modes with the following inputs:

x = horizontal position [i.e., (0,639) if VGA_HI]
y1 = start point in the y-axis [(0,479) if VGA_HI]
y2 = end point in the y-axis [(0,479) if VGA_HI]
color# = line's color (1,15)
(See "How to read the toolbox" for color codes)

Examples:

VGA_HI 100 20 300 12 VERT_LINE

See also:

HORIZ_LINE_BIOS, HORIZ_LINE, VERT_LINE_BIOS, LINE

VERT_LINE_BIOS (x y1 y2 color --)

Type: WORD

Category: Video BIOS call drawing routine

Purpose:

Provide a vertical (slope=infinite) straight line drawing capability

Description:

This routine draws a vertical line in EGA/VGA graphics modes with the following inputs:

x = horizontal position [i.e., (0,639) if VGA_HI]
y1 = start point in the y-axis [(0,479) if VGA_HI]
y2 = end point in the y-axis [(0,479) if VGA_HI]
color# = line's color (1,15)
(See "How to read the toolbox" for color codes)

Examples:

VGA_HI 100 20 300 12 VERT_LINE_BIOS

See also:

HORIZ_LINE_BIOS, HORIZ_LINE, VERT_LINE_BIOS, LINE

VGA_HI (--)

Type: WORD

Category: Video screen environment setup routine

Purpose:

Set the system to the specified graphics mode

Description:

Invoking VGA_HI will bring the screen to high-resolution-EGA mode by calling SET_GRAPHMODE, a video BIOS function routine. This call corresponds to setting the video to:

640x480 16-color graphics

It also sets up the graphics character heights to be 16 units (pixels) high. (EGA_HI has 14 units high character size)

During the process, the screen gets cleared.

Examples:

VGA_HI 50 50 200 200 12 LINE_BIOS

See also:

CGA_HI, CGA_LO, EGA_LO, EGA_HI, VGA_LO

WRITE_CHAR (char --)

Type: CODE

Category: Video BIOS function call 9 and 10

Purpose:

Write character at current cursor position

Description:

WRITE_CHAR is a video BIOS call routine that writes a string of characters at current cursor position.

Input:

character

Examples:

abc WRITE_CHAR

See also:

WRITE_TCHAR

WRITE_TCHAR (char --)

Type: CODE

Category: Video BIOS function call 14

Purpose:

Write character at current cursor position and move the cursor to next position.

Description:

WRITE_CHAR is a video BIOS call routine which writes character at current cursor position and the cursor is moved to the next position. Unlike the other write character functions, this function interprets the bell, carriage return, and linefeed characters as commands rather than characters from the IBM set.

Input:

character

Examples:

abc WRITE_TCHAR

See also:

WRITE_CHAR

XOR_VIDEO (--)

Type: WORD

Category: video environment controlling routine

Purpose:

Specify functions (AND, OR, XOR) available for updating pixels during pixel write modes.

Description:

XOR_VIDEO specifies the Data Rotate/Function Select register's (03H) two bit fields to 00011000. This bit pattern forces latched pixels to be XORed when updated. Note that the variable PIXEL_MODE stores the two bit fields.

Examples:

VGA_HI XOR_VIDEO 250 300 12 PUT_PIXEGA

See also:

AND_VIDEO, OR_VIDEO

APPENDIX B

FORTH GRAPHICS TOOLBOX
SOURCE CODE

```
\ GRAPHICS.SEQ    GRAPHICS PACKAGE FOR RFF   9/90  K. YAN
\ REVISED   3/91  H. KO
HEAPOFF
TOMODE OFF
\ ***************** VIDEO BIOS FUNCTION CALLS  0 - 15 ********************
\ ********* SCREEN ENVIRONEMENT ( MODE, CURSOR POSITION) ***************
HEX F/H
CODE FONTAD ( -- seg os )
        POP BX              PUSH BP
        MOV AX, # 1130
        INT # 010
        MOV AX, BP          POP BP
        PUSH ES             PUSH AX       \ ES - SEG, AX - OFS
        NEXT                END-CODE
H/F
CODE SET_GRAPHMODE ( graphmode# -- )
\ function call 0
\ Sets the system to the specified graphics mode as shown below:
\       0 & 1 - 40 column alphanumeric (CGA compatible)
\       2 & 3 - 80 column alphanumeric (CGA compatible)
\       4 & 5 - 320x200 4-color graphics limited to 2 palettes (CGALo)
\           6 - 640x200 2-color graphics: one must be black (CGAHi)
\           7 - monochrome alphanumeric (monochrome adapter compatible)
\       8- 12 - reserved
\          13 - 320x200 16 color
\          14 - 640x200 16 color (EGALo)
\          15 - 640x350 monochrome graphics (EGAMonoHi)
\          16 - 640x350 16 color (EGAHi)
\          17 - 640x480 monochrome graphics (VGAMonoHi)
\          18 - 640x480 16 color (VGAHi)
\          19 - 320x200 256 color (VGA only)
\
        POP AX
        INT # 010
        NEXT            END-CODE

CODE GOTO_XY     ( x y -- )
\  function call 2 ( also known as SETCURPOS )
\  Give the coordinates for the cursor's position on the screen.
\  y=row 0 is the top of the screen, and x=column 0 is the left side
\  of the screen.
        POP AX      POP DX
        MOV DH, AL                  ( DH - y position )
        XOR BX, BX                  ( DL - x position )
        MOV AX, # 200
        INT # 10
        NEXT            END-CODE

 CODE GET_XY ( -- x y )
\  function call 3  ( also known as RDCURPOS )
\  Get current (x,y) cursor position

        MOV AX, # 300  XOR BX, BX
        INT # 10
        MOV AL, DH      XOR AH, AH      ( DH - y LOCATION )
        XOR DH, DH                      ( DL - x LOCATION )
        PUSH DX         PUSH AX
        NEXT            END-CODE

\  function call 4:  READ LIGHT PEN POSITION
\  VGA does not support a light pen

 CODE SET_ACTIVEPAGE ( page# -- )
\  function call 5
\  Selects Page as the active page for graphics output.
\  The adapter may have several pages (or screens) of information in
```



```
\  memory.  Only one page is visible at any one time- this is called
\  active display page.  Most of the functions which allow you to modify
\  the screen (write characters, plot points, move the cursor, etc.) also
\  let you choose which page to modify and thus an invisible screen may be
\  changed.  Through this feature, you may display one page while another
\  is being created, and then immediately switch to the new screen ( a
\  technique useful for animation or slide shows).  This function lets you
\  choose which screen is displayed.  Usually, screen 0 is the only screen
\  displayed and modified.

        POP AX                  ( TEXT = 0-7            )
        MOV AH, # 05            ( EGA  = 0-1            )
        INT # 010               ( VGA  = 0              )
        NEXT            END-CODE

CODE SCROLL_PAGEUP  ( #oflines -- )
\  function call 6
\  This function scrolls the text on the screen-lines move from
\  the bottom of the screen toward the top, and blank lines are inserted
\  at the bottom.  Note that corners of a window can be specified, so
\  that only a portion of the screen scrolls.  Register AL is set to the
\  number of lines to scroll;

        POP AX
        MOV AH, # 06
        XOR BH, BH
        XOR CX, CX
        MOV DX, # FFFF
        INT # 010
        NEXT            END-CODE

CODE SCROLL_PAGEDOWN  ( #oflines -- )
\  function call 7
\  this function scrolls the text on the screen - lines move from the top
\  of the screen toward the bottom, and blank lines are inserted at the top.

        POP AX
        MOV AH, # 07
        XOR BH, BH
        XOR CX, CX          ( upper left corner 0h,0h of a window )
        MOV DX, # FFFF      ( lower right corner FFh,FFh of a window )
        INT # 010
        NEXT            END-CODE


CODE READ_CHAR ( -- char )
\  function call 8
\  Read character at current cursor position.

        MOV AX, # 0800   XOR BH, BH
        INT # 010
        XOR AH, AH       PUSH AX
        NEXT             END-CODE

CODE WRITE_CHAR ( char -- )
\  function call 9 & 10
\  Write character at current cursor position.

        POP AX
        MOV AH, # 0A00   XOR BH, BH
        MOV CX, # 1
        INT # 010
        NEXT            END-CODE

CODE PUT_PIXEL ( x y color# -- )
\  function call 12
```

```
\ This function call is used to plot a point on the screen.
\ It sets the pixel at (x,y) to the color# (0 - 15) specified.

      POP AX     POP DX     POP CX
       XOR BH, BH      OR AX, # 0C00
       INT # 010
       NEXT           END-CODE

CODE GET_PIXEL ( x y -- color# )
\ function call 13
\ Reads the color# of the pixel at (x,y)

       POP DX          POP CX
       MOV AH, # 0D00   XOR BH, BH
       INT # 010
       XOR AH, AH      PUSH AX
       NEXT            END-CODE

CODE WRITE_TCHAR ( char -- )
\ function call 14
\ A character is written and the cursor is moved to the next position.
\ Unlike the other write character functions, this function interprets
\ the bell, carriage return, and linefeed characters as commands rather
\ than characters from the IBM set.

       POP AX
       MOV AH, # 0E00
       XOR BX, BX      MOV BL, # 7
       INT # 010
       NEXT            END-CODE

CODE GET_GRAPHMODE ( -- video mode )
\ function call 15
\ Returns information about the current graphics mode setting.

       MOV AX, # 0F00
       INT # 010
       XOR AH, AH              ( VIDEO MODE # = AL      )
       PUSH AX
       NEXT           END-CODE

CODE SET_PALETTE ( palette# color# -- )
\ function 16
\ Changes the setting of the color number entry in the palette
\ to new color

       POP CX          POP BX
       MOV BH, CL      MOV AX, # 01000
       INT # 010
       NEXT            END-CODE

CODE SET_BORDER ( color# -- )

\ function call 16
\ Subservice call 01h  VGA/EGA graphics only
\ Changes the setting of the color number entry in the palette
\ to new color

       POP CX
       MOV BH, CL      MOV AX, # 01001
       INT # 010
       NEXT            END-CODE


CODE GEN_CHAR ( code# -- )
```



```
\ function call 17
\ Subservice call 01h  VGA/EGA graphics only
\ Changes the setting of the color number entry in the palette
\ to new color

       POP CX
       MOV AL, CL      MOV AX, # 01100
       INT # 010
       NEXT            END-CODE

\ ********** EGA/VGA DIRECT VIDEO ACCESS ROUTINES **********

VARIABLE PIXEL_MODE
VARIABLE DIRECT_VIDEO

DIRECT_VIDEO ON

F/H
LABEL EGAPIX_ADDR     ( CALCULATE EGA/VGA PIXEL ADDRESS )
       MOV CL, BL

       PUSH DX
       MOV DX, # 050            ( # OF BYTES PER LINE    )
       MUL DX
       POP DX

       SHR BX, 1       SHR BX, 1        SHR BX, 1
       ADD BX, AX
       MOV AX, # 0A000      ( EGA/VGA GRAPHICS MEMORY ADDRESS = A000 )
       MOV ES, AX
       AND CL, # 7             ( BIT MASK = 7                    )
       XOR CL, # 7
       MOV AH, # 1
       RET             END-CODE

LABEL EGA_RESET                ( RESET EGA VIDEO REGISTERS )
       MOV AX, # FF08          ( RESET BIT MASK = FF, ALL BITS  )
       OUT DX, AX
       MOV AX, # 0005            ( RESET READ/WRITE MODE = 0 )
       OUT DX, AX
       RET             END-CODE

H/F
CODE PUT_PIXEGA ( x y color#  -- )
       POP CX   POP AX   POP BX
       PUSH CX
       CALL EGAPIX_ADDR
       SHL AH, CL                  ( SET BIT MASK            )
       MOV AL, # 8                 ( BIT MASK ACCESS = 8     )
       MOV DX, # 03CE              ( GRAPHICS CONTROLLER REG )
       OUT DX, AX
       MOV AX, # 0205              ( WRITE MODE = 2 )
       OUT DX, AX
       MOV AX, PIXEL_MODE
       MOV AH, AL
       MOV AL, # 3                 ( DATA ROTATE/FUNCTION SELECT REG     )
       OUT DX, AX                  ( 0003 - WRITE W/O MODIFICATION       )
                                 ( 0803 - AND DATA WITH LATCH CONTENTS )
                                 ( 1003 - OR DATA WITH LATCH CONTENTS  )
                                 ( 1803 - XOR DATA WITH LATCH CONTENTS )
       MOV AL, ES: [BX]            ( READ PIXEL TO LATCH )
       POP AX                      ( GET PIXEL COLOR     )
       MOV ES: [BX], AL            ( WRITE PIXEL         )
       CALL EGA_RESET
       NEXT                END-CODE
```

```
CODE GET_PIXEGA ( x y -- color )
        POP AX          POP BX   (  AX -> y, BX -> x )
        CALL EGAPIX_ADDR
        MOV CH, AH
        SHL CH, CL
        PUSH SI
        MOV SI, BX
        XOR BL, BL
        MOV DX, # 03CE
        MOV AX, # 0304
        BEGIN                           ( LOOP THRU ALL 4 VIDEO PLANES )
                OUT DX, AX                 ( SELECT BIT PLANE            )
                MOV BH, ES: [SI]           ( BH-> BIT PLANE VALUE        )
                AND BH, CH                 ( MASK ONE BIT                )
                NEG BH                     ( BIT = 1 IF MASK = 1 ELSE = 0 )
                ROL BX, # 1                ( BL -> NEXT BIT PLANE VALUE  )
                DEC AH                     ( NEXT BIT PLANE VALUE        )
        < UNTIL
        POP SI
        MOV AL, BL
        XOR AH, AH
        PUSH AX                    ( AL -> COLOR  )
        CALL EGA_RESET
        NEXT            END-CODE

CODE SET_VPLANE ( Hex# -- )         ( ENABLE/DISABLE VIDEO PLANE    )
        POP AX                        ( WORKS IN VIDEO MODE = 0 ONLY )
        MOV AH, AL                    ( BIT  0- PLANE 0              )
        MOV AL, # 2                   (      1- PLANE 1              )
        MOV DX, # 03C4                (      2- PLANE 2              )
        OUT DX, AX                    (      3- PLANE 3              )
        NEXT            END-CODE      ( 1-> ENABLE, 0-> DISABLE      )

\ ***************** DRAW VERTICAL LINE ********************

CODE VERT_LINE  ( x y1 y2 color -- )
        POP BX                  \ GET COLOR
        MOV DX, # 03CE          \ SET UP GRAPHICS CONTROLLER
        MOV AH, BL              \ USE WRITE MODE 0
        XOR AL, AL              \ ENABLE ONLY PLANES AFFECTED
        OUT DX, AX
        MOV AX, # 0F01          \ ENABLE PLANES
        OUT DX, AX
        MOV AH, PIXEL_MODE      \ PIXEL WRITE MODE
        MOV AL, # 3
        OUT DX, AX
        POP BX    POP AX        \ y2, y1
        MOV CX, BX
        SUB CX, AX              \ Y = Y2-Y1
       < IF
                NEG CX
                MOV AX, BX
        THEN
        INC CX
        POP BX                  \ GET X LOCATION
        PUSH CX                 \ SAVE COUNTER = Y -> Y2-Y1
        CALL EGAPIX_ADDR
        SHL AH, CL              \ MASK BITS IN BYTE
        MOV AL, # 8
        OUT DX, AX
        POP CX
        DEC CX
        PUSH SI
        MOV SI, # 050           \ HEX 50 = 80 BYTES/LINE
        BEGIN                   \ DRAW LINE LOOP
                OR  ES: [BX], AL      \ CX IS y COUNTER
```



```
                ADD BX, SI
                DEC CX
        < UNTIL
        POP SI
        XOR AX, AX              \ RESET VIDEO REGISTERS
        OUT DX, AX
        INC AX
        OUT DX, AX
        MOV AL, # 3
        OUT DX, AX
        MOV AX, # 0FF08
        OUT DX, AX
        NEXT            END-CODE

\ ************* DRAW HORIZONTAL LINE **************

CODE HORIZ_LINE ( y x1 x2 color -- )
        PUSHIP
        SUB BP, # 4
        POP BX                          \ GET COLOR
        MOV DX, # 03CE                  \ SET UP GRAPHICS CONTROLLER
        MOV AH, BL                      \ USE WRITE MODE 0
        XOR AL, AL                      \ ENABLE ONLY PLANES AFFECTED
        OUT DX, AX
        MOV AX, # 0F01                  \ ENABLE PLANES
        OUT DX, AX
        MOV AH, PIXEL_MODE              \ PIXEL WRITE MODE
        MOV AL, # 3
        OUT DX, AX
        POP CX    POP BX     POP AX     \ X2, X1, Y
        MOV 0 [BP], BX                  \ 0 [BP] = X1
        MOV 2 [BP], CX                  \ 2 [BP] = X2
        SUB CX, BX                      \ FORCE X1 < X2
        < IF
                MOV BX, 2 [BP]
                XCHG BX, 0 [BP]
                MOV 2 [BP], BX
                MOV BX, 0 [BP]
        THEN
        CALL EGAPIX_ADDR
        MOV DI, BX                      \ ES:DI -> BUFFER
        MOV DH, AH                      \ DH = UNSHIFTED BIT MASK FOR LEFTMOST BYTE
        NOT DH
        SHL DH, CL                      \ DH = REVERSE BIT MASK FOR FIRST BYTE
        NOT DH                          \ DH : BIT MASK FOR FIRST BYTE
        MOV CX, 2 [BP]                  \ CX = X2
        AND CL, # 7
        XOR CL, # 7                     \ CL = NUMBER OF BITS TO SHIFT LEFT
        MOV DL, # OFF                   \ DL = UNSHIFTED BIT MASK FOR RIGHTMOST BYTE

        SHL DL, CL                      \ DL = BIT MASK FOR LAST BYTE
        MOV AX, 2 [BP]                  \ AX = X2
        MOV BX, 0 [BP]                  \ BX = X1
        MOV CL, # 3                     \ NUMBER OF BITS TO SHIFT TO CONVERT PIXELS
                                        \ TO BYTE
        SHR AX, CL                      \ BYTE OFFSET OF X2
        SHR BX, CL                      \ BYTE OFFSET OF X1
        MOV CX, AX
        SUB CX, BX                      \ CX = NUMBER OF BYTES IN LINE - 1
        MOV BX, DX                      \ BH = BIT MASK FOR FIRST BYTE
        MOV DX, # 03CE                  \ BL = BIT MASK FOR LAST BYTE
        MOV AL, # 8                     \ DX = GRAPHICS CONTROLLER PORT
                                        \ AL = BIT MASK REGISTER
        PUSH ES
        POP DS                          \ VIDEO BUFFER -> DS:SI
        MOV SI, DI
```

```
        OR BH, BH                       \ JUMP IF BYTE ALIGNED ( X1 IS LEFTMOST
        0>= IF                          \   PIXEL IN BYTE )
                OR CX, CX               \ JUMP IF MORE THAN ONE BYTE IN LINE
                0= IF
                        AND BL, BH      \ BL = BIT MASK FOR THE LINE
                ELSE
                        MOV AH, BH      \ AH = BIT MASK FOR FIRST BYTE
                        OUT DX, AX      \ UNDATE GRAPHICS CONTROLLER
                        MOVSB           \ UNDATE BIT PLANES
                        DEC CX
                        MOV AH, # OFF   \ AH = BIT MASK
                        OUT DX, AX      \ UPDATE BIT MASK
                        REP MOVSB       \ UPDATE ALL PIXELS IN LINE
                THEN
        ELSE
                MOV AH, # FF
                OUT DX, AX
                REP MOVSB
        THEN
        MOV AH, BL                      \ AH = BIT MASK FOR LAST BYTE
        OUT DX, AX                      \ UPDATE GRAPHICS CONTROLLER
        MOVSB                           \ UPDATE BIT PLANES
        XOR AX, AX                  \ ****** RESET VIDEO REGISTERS ******
        OUT DX, AX                  \ RESTORE SET/RESET REGISTER
        INC AX
        OUT DX, AX
        MOV AL, # 3                 \ RESTORE DATA ROTATE/FUNC SELECT REGISTER
        OUT DX, AX
        MOV AX, # 0FF08             \ RESTORE BIT MASK REGISTER
        OUT DX, AX
        ADD BP, # 4
        POPIP
        NEXT            END-CODE
\ ************** RESET REGISTERS AFTER LINE ROUTINE ***************
LABEL EXIT_LINE
        XOR AX, AX
        OUT DX, AX                  \ RESTORE SET/RESET REGISTER
        INC AX
        OUT DX, AX                  \ RESTORE ENABLE SET/RESET REGISTER
        MOV AL, # 3
        OUT DX, AX                  \ RESTORE DATA ROTATE/FUNC SELECT REGISTER
        MOV AX, # 0FF08
        OUT DX, AX                  \ RESTORE BIT MASK REGISTER
        RET             END-CODE

\ ************ LINE ROUTINE FOR SLOPE <= 1 ************
F/W
CODE LS_LINE  ( x1 y1 x2 y2 color -- )
  REVEAL                        \ ALLOW EMBEDDED LABELS
        PUSHIP
        SUB BP, # E             \ MAKE SPACE FOR 7 TEMP VAR
        POP BX                      \ GET COLOR
        MOV DX, # 03CE              \ SET UP GRAPHICS CONTROLLER
        MOV AH, BL                  \ USE WRITE MODE 0
        XOR AL, AL                  \ ENABLE ONLY PLANES AFFECTED
        OUT DX, AX
        MOV AX, # 0F01              \ ENABLE PLANES
        OUT DX, AX
        MOV AH, PIXEL_MODE          \ PIXEL WRITE MODE
        MOV AL, # 3
        OUT DX, AX
        MOV SI, # 050
        POP AX    POP CX    POP BX    POP DX  \ Y2, X2, Y1, X1
        MOV 0 [BP], DX          \ X1  TEMPORARY VARIABLES
        MOV 2 [BP], BX          \ Y1
        MOV 4 [BP], CX          \ X2 .
```



```
        MOV 6 [BP], AX          \ Y2
        SUB CX, DX              \ CX = dx = X2 - X1
        < IF                    \ FORCE X1 < X2
                NEG CX
                MOV BX, 4 [BP]          \ EXCHANGE X1 - X2
                XCHG BX, 0 [BP]
                MOV 4 [BP], BX
                MOV BX, 6 [BP]          \ EXCHANGE Y1 - Y2
                XCHG BX, 2 [BP]
                MOV 6 [BP], BX
        THEN
        MOV BX, 6 [BP]
        SUB BX, 2 [BP]          \ BX = Y2 - Y1
        < IF                    \ CHECK FOR POSITIVE SLOPE
                NEG BX              \ BX = Y1 - Y2
                NEG SI              \ NEGATE INCREMENT FOR BUFFER INTERLEAVE
        THEN
        MOV 8 [BP], SI              \ BP+8 -> VAR_VERT_INCR
\       CMP BX, CX
        SHL BX, # 1                 \ BX -> 2 * dy
  DECIMAL
        MOV 10 [BP], BX             \ BP+10 -> INCR1 = 2 * dy
        SUB BX, CX
        MOV SI, BX                  \ SI = d = 2 * dy - dx
        SUB BX, CX
        MOV 12 [BP], BX             \ BP+12 -> INCR2 = 2 * ( dy - dx )
  HEX
        PUSH CX
        MOV AX, 2 [BP]              \ AX -> Y1
        MOV BX, 0 [BP]              \ BX -> X1
        CALL EGAPIX_ADDR
        MOV DI, BX                  \ ES:DI -> BUFFER
        SHL AH, CL
        MOV BL, AH                  \ BL = BIT MASK FOR PIXEL
        MOV AL, # 8                 \ AL = BIT MASK REGISTER NUMBER
        POP CX
        INC CX
        MOV DX, # 03CE              \ GRAPHICS CONTROLLER PORT
\ ************************ SLOPE OF LINE <= 1 ************************
LABEL L10
        MOV AH, BL                  \ AH = BIT MASK OF NEXT PIXEL
LABEL L11
        OR  AH, BL                  \ MASK CURRENT PIXEL
        ROR BL, # 1                 \ ROTATE PIXEL VALUE
        NC IF                       \ JUMP IF LEFTMOST PIXEL LOCATION
                OR  SI, SI              \ TEST SIGN OF d
                < IF
  DECIMAL
                        ADD SI, 10 [BP]     \ d = d + INCR1
  HEX
                        DEC CX
                        > IF   JMP L11
                        THEN
                        OUT DX, AX          \ UPDATE BIT MASK REGISTER
                        OR  ES: [DI], AL    \ UPDATE BIT PLANES
                        CALL EXIT_LINE
                        ADD BP, # E
                        POPIP
                        NEXT
                THEN
  DECIMAL
                ADD SI, 12 [BP]         \ d = d + INCR2
  HEX
                OUT DX, AX              \ UPDATE BIT MASK REGISTER
                OR ES: [DI], AL         \ UPDATE BIT PLANES
                ADD DI, 8 [BP]          \ INCREMENT y
```

```
            DEC CX
            > IF    JMP L10
            THEN
            CALL EXIT_LINE
            ADD BP, #^E
            POPIP
            NEXT
      THEN
      OUT DX, AX                    \ UPDATE BIT MASK REGISTER
      OR ES: [DI], AL               \ UPDATE BIT PLANES
      INC DI                        \ INCREMENT x
      OR  SI, SI                    \ TEST SIGN OF d
      < IF
DECIMAL
            ADD SI, 10 [BP]         \ d = d + INCR1
HEX
            DEC CX
            > IF    JMP L10
            THEN
            CALL EXIT_LINE
            ADD BP, #^E
            POPIP
            NEXT
      THEN
DECIMAL
      ADD SI, 12 [BP]               \ d = d + INCR2
      ADD DI, 8 [BP]                \ VERTICAL INCREMENT
HEX
      DEC CX
     > IF     JMP L10
     THEN
     CALL EXIT_LINE
     ADD BP, #^E
     POPIP
     NEXT       END-CODE

\ ************** LINE ROUTINE FOR SLOPE > 1 **************
CODE HS_LINE ( x1 y1 x2 y2 color -- )
  REVEAL                        \ ALLOW EMBEDDED LABELS

      PUSHIP
      SUB BP, # 0E              \ MAKE SPACE FOR 7 TEMP VAR
      POP BX                        \ GET COLOR
      MOV DX, # 03CE                \ SET UP GRAPHICS CONTROLLER
      MOV AH, BL                    \ USE WRITE MODE 0
      XOR AL, AL                    \ ENABLE ONLY PLANES AFFECTED
      OUT DX, AX
      MOV AX, # 0F01                \ ENABLE PLANES
      OUT DX, AX
      MOV AH, PIXEL_MODE            \ PIXEL WRITE MODE
      MOV AL, # 3
      OUT DX, AX
      MOV SI, # 050
      POP AX    POP CX    POP BX    POP DX  \ Y2, X2, Y1, X1
      MOV 0 [BP], DX          \ X1  TEMPORARY VARIABLES
      MOV 2 [BP], BX          \ Y1
      MOV 4 [BP], CX          \ X2
      MOV 6 [BP], AX          \ Y2
      SUB CX, DX              \ CX = dx = X2 - X1
      < IF                    \ FORCE X1 < X2
            NEG CX
            MOV BX, 4 [BP]          \ EXCHANGE X1 - X2
            XCHG BX, 0 [BP]
            MOV 4 [BP], BX
            MOV BX, 6 [BP]          \ EXCHANGE Y1 - Y2
            XCHG BX, 2 [BP]
```



```
            MOV 6 [BP], BX
      THEN
      MOV BX, 6 [BP]
      SUB BX, 2 [BP]        \ BX = Y2 - Y1
      < IF                  \ CHECK FOR POSITIVE SLOPE
            NEG BX                  \ BX = Y1 - Y2
            NEG SI                  \ NEGATE INCREMENT FOR BUFFER INTERLEAVE
      THEN
      MOV 8 [BP], SI            \ BP+8 -> VAR_VERT_INCR
      XCHG BX, CX               \ ***** EXCHANGE dx, dy ********
      SHL BX, # 1               \ BX -> 2 * dy
DECIMAL
      MOV 10 [BP], BX           \ BP+10 -> INCR1 = 2 * dy
      SUB BX, CX
      MOV SI, BX                \ SI = d = 2 * dy - dx
      SUB BX, CX
      MOV 12 [BP], BX           \ BP+12 -> INCR2 = 2 * ( dy - dx )
HEX
      PUSH CX
      MOV AX, 2 [BP]            \ AX -> Y1
      MOV BX, 0 [BP]            \ BX -> X1
      CALL EGAPIX_ADDR
      MOV DI, BX                \ ES:DI -> BUFFER
      SHL AH, CL
      MOV BL, AH                \ BL = BIT MASK FOR PIXEL
      MOV AL, # 8               \ AL = BIT MASK REGISTER NUMBER
      POP CX
      INC CX
      MOV DX, # 03CE            \ GRAPHICS CONTROLLER PORT
\ **************  SLOPE OF LINE > 1 **************
      MOV BX, 8 [BP]            \ BX = y - INCREMENT
LABEL L21
      OUT DX, AX                \ UPDATE BIT MASK REGISTER
      OR  ES: [DI], AL          \ UPDATE BIT PLANES
      ADD DI, BX                \ INCREMENT y
      OR  SI, SI                \ TEST SIGN OF d
      < IF
DECIMAL
            ADD SI, 10 [BP]         \ d = d + INCR1
HEX
            DEC CX
            > IF    JMP L21
            THEN
            CALL EXIT_LINE
            ADD BP, #^0E
            POPIP
            NEXT
      THEN
DECIMAL
      ADD SI, 12 [BP]           \ d = d + INCR2
HEX
      ROR AH, # 1               \ ROTATE BIT MASK
      ADC DI, # 0               \ INCREMENT DI IF LEFTMOST PIXEL POSITION
      DEC CX
      > IF    JMP L21
      THEN
      CALL EXIT_LINE
      ADD BP, #^0E
      POPIP
      NEXT          END-CODE

DECIMAL

LABEL LONGMULTIPLY      ( this subroutine used in CODE ELLIPSE        )
                        ( caller:   DX = u1 [ hi-order word of 32-bit )
                        (               number                        )
```

```
                        (          AX = u2 [ lo-order word  ]       )
                        (          CX = v1 [ 16-bit number  ]       )
                        ( returns:  DX:AX = 32-bit result           )

                PUSH AX         ( preserve u2  )
                MOV AX, DX      ( AX=u1 )
                MUL CX          ( AX= hi-order word of result )
                XCHG AX, CX     ( AX= v1, CX= hi-order word   )
                POP DX          ( DX = u2  )
                MUL DX          ( AX = lo-order word of result )
                                ( DX = carry  )

                ADD DX, CX      ( CX= hi-order word of result  )

        RET     END-CODE


LABEL SET4PIXELS        ( this subroutine used in CODE ELLIPSE )
                        ( call with CH= y- increment [0,-1] )
                        (           CL= x- increment [0,1]  )

        PUSH AX         ( preserve these regs )
        PUSH BX
        PUSH DX
HEX
        MOV DX, # 03CE  ( DX= graphics controller port )

        XOR BX, BX      ( BX = 0  )
        TEST CH, CH
      0<> IF            ( jump if y-increment =0   )

DECIMAL
        MOV BX, # 0080  ( BX= positive increment )
        NEG BX          ( BX= negative increment )
        THEN

        MOV AL, # 8     ( AL = Bit Mask reg number )

\ pixels at (xc-x, yc+y) and (xc-x, yc-y)

        XOR SI, SI      ( SI = 0 )

        MOV AH, -10 [BP]

        ROL AH, CL      ( AH = bit mask rotated horizontally )
        RCL SI, # 0001  ( SI = 1 if bit mask rotated around )
        NEG SI          ( SI = 0 or -1  )

        MOV DI, SI      ( SI,DI = left horizontal increment )

        ADD SI, -2 [BP] ( SI = upper left addr + horiz incr  )
        ADD SI, BX      ( SI = new upper left addr  )
        ADD DI, -6 [BP]
        SUB DI, BX      ( DI = new lower left addr  )

        MOV -10 [BP], AH   ( update these variable )
        MOV -2 [BP], SI
        MOV -6 [BP], DI
        OUT DX, AX      ( update Bit Mask register )

        MOV CH, ES: [SI]   ( update upper left pixel )
        MOV ES: [SI], CH   ( update lower left pixel )
        MOV CH, ES: [DI]
        MOV ES: [DI], CH

\ pixels at (xc-x, yc+y) and (xc-x, yc-y)
```



```
        XOR SI, SI      ( SI = 0 )
        MOV AH, -12 [BP]

        ROR AH, CL      ( AH = bit mask rotated horizontally )
        RCL SI, # 0001  ( SI = 1 if bit mask rotated around  )

        MOV DI, SI      ( SI,DI = right horizontal increment )

        ADD SI, -4 [BP]  ( SI = upper right addr + horiz incr  )
        ADD SI, BX       ( SI = new upper right addr   )
        ADD DI,  -8 [BP]
        SUB DI, BX       ( DI = new lower right addr  )

        MOV -12 [BP], AH  ( update these variables  )
        MOV -4 [BP], SI
        MOV -8 [BP], DI

        OUT DX, AX       ( update Bit Mask register  )

        MOV CH, ES: [SI]  ( update upper right pixel  )
        MOV ES: [SI], CH
        MOV CH, ES: [DI]  ( update lower right pixel  )
        MOV ES: [DI], CH

        POP DX            ( restore these regs  )
        POP BX
        POP AX

  RET   END-CODE

CODE ELLIPSE  ( color# , b, a, yc, xc )

\ input: color#, b=minor axis, a= major axis, yc&xc= center of ellipse
\
\ function:  draw an ellipse in EGA/VGA graphics modes.


 REVEAL
\ initialize registers
        PUSHIP          ( preserve  IP information )
        PUSH BP         ( preserve caller registers )
        MOV BP, SP
        SUB SP,  # 0040  ( reserve local stack space, 40 bytes )
        PUSH SI
        PUSH DI

\ set graphics controller mode register
 HEX
        MOV DX, # 03CE   ( DX = graphics controller I/O port )
        MOV AX, # 0005   ( AL = mode register number )
        OUT DX, AX       ( AH = write mode 0 [bits 0,1], read mode 0 [bit 4])

\  set data rotate/funtion select register

        MOV AH, # 0      ( AH = read-modify-write bits )
        MOV AL, # 3      ( AL = data rotate/function select register )
        OUT DX, AX

\  set set/reset and enable set/reset registers
DECIMAL


        MOV AH, 10 [BP]  ( AH = color#                    )    \ ARGcolor#
        MOV AL, # 0      ( AL = set/reset register number )
        OUT DX, AX
```

```
HEX
        MOV AX, # 0F01   ( AH = value for enable set/reset )
                         ( [all bit planes enabled]        )
        OUT DX, AX       ( AL = enable set/reset reg number)

\ Initial constants
 DECIMAL

        MOV AX, 6 [BP]                                          \ ARGa
        MUL AX
        MOV -28 [BP], AX
        MOV -30 [BP], DX  ( a^2 )
        SHL AX, # 0001
        RCL DX, # 0001
        MOV -36 [BP], AX
        MOV -38 [BP], DX  ( 2*a^2 )


        MOV AX, 8 [BP]                                          \ ARGb
        MUL AX
        MOV -32 [BP], AX
        MOV -34 [BP], DX  ( b^2 )
        SHL AX, # 0001
        RCL DX, # 0001
        MOV -40 [BP], AX
        MOV -42 [BP], DX  ( 2*b^2 )

\ plot pixels from (0,b) until dy/dx = -1
\ initial buffer address and bit mask

        MOV AX, # 080     ( AL = video buffer line length )
        MUL 8 [BP]        ( AX = relative byte offset of b )    \ ARGb
        MOV SI, AX
        MOV DI, AX

        MOV AX, 4 [BP]    ( AX = yc )                           \ ARGyc
        MOV BX, 2 [BP]    ( BX = xc )                           \ ARGxc
        CALL EGAPIX_ADDR  ( AH = bit mask                )
                          ( ES:BX -> buffer              )
                          ( CL = #bits to shift left     )

        MOV AH, # 1       ( AH = bit mask for first pixel  )
        SHL AH, CL
        MOV -10 [BP], AH
        MOV -12 [BP], AH

        ADD SI, BX        ( SI = offset of [0,b]  )
        MOV -2 [BP], SI
        MOV -4 [BP], SI
        SUB BX, DI        ( BX = offset of [0,-b] )
        MOV -6 [BP], BX
        MOV -8 [BP], BX

\ initial decision variables

        XOR AX, AX
        MOV -20 [BP], AX
        MOV -22 [BP], AX  ( dx = 0 )

        MOV AX, -36 [BP]
        MOV DX, -38 [BP]
        MOV CX, 8 [BP]                                          \ ARGb
        CALL LONGMULTIPLY   ( perform 32-bit by 16-bit multiply )
        MOV -24 [BP], AX
        MOV -26 [BP], DX    ( dy = 2*a^2 * b )
```



```
        MOV AX, -28 [BP]
        MOV DX, -30 [BP]    ( DX:AX = a^2 )
        SAR DX, # 0001
        RCR AX, # 0001
        SAR DX, # 0001
        RCR AX, # 0001      ( DX:AX = a^2 /4 )

        ADD AX, -32 [BP]
        ADC DX, -34 [BP]    ( DX:AX = b^2 + a^2 /4  )
        MOV -16 [BP], AX
        MOV -18 [BP], DX

        MOV AX, -28 [BP]
        MOV DX, -30 [BP]
        MOV CX, 8 [BP]                                          \ ARGb
        CALL LONGMULTIPLY   ( DX:AX = a^2 * b )
        SUB -16 [BP], AX
        SBB -18 [BP], DX    ( d = b^2 - a^2 + a^2 /4 )

\ loop until dydx >= -1

        MOV BX, 8 [BP]      ( BX= inital y-coordinate )         \ ARGb
        XOR CX, CX          ( CH = 0    [ initial y-increment ] )
                            ( CL = 0    [ initial x-increment ] )
    BEGIN

        MOV AX, -20 [BP]
        MOV DX, -22 [BP]
        SUB AX, -24 [BP]
        SBB DX, -26 [BP]

       0< WHILE             ( jump if dx>=dy  )

        CALL SET4PIXELS

        MOV CX, # 0001      ( CH = 0  [y-increment] )
                            ( CL = 1  [x-increment] )
        CMP -18 [BP], # 0000

       0>= IF               ( jump if d < 0 )

        MOV CH, # 1         ( increment in y direction  )
        DEC BX              ( decrement current y-coordinate )

        MOV AX, -24 [BP]
        MOV DX, -26 [BP]
        SUB AX, -36 [BP]
        SBB DX, -38 [BP]    ( DX:AX = dy- 2*a^2 )
        MOV -24 [BP], AX
        MOV -26 [BP], DX    ( dy -= 2*a^2 )

        SUB -16 [BP], AX
        SBB -18 [BP], DX    ( d -= dy )

        ELSE

        MOV AX, -20 [BP]
        MOV DX, -22 [BP]
        ADD AX, -40 [BP]
        ADC DX, -42 [BP]    ( DX:AX = dx + 2*b^2  )
        MOV -20 [BP], AX
        MOV -22 [BP], DX    ( dx += 2*b^2  )

        ADD AX, -32 [BP]
        ADC DX, -34 [BP]    ( DX:AX = dx + b^2  )
```

```
        ADD -16 [BP], AX
        ADC -18 [BP], DX    ( d += dx + b^2  )
        THEN

        REPEAT

\    plot pixels from current (x,y) until y < 0
\    initial buffer address and bit mask

        PUSH BX             ( preserve current y-coordinate )
        PUSH CX             ( preserve x- and y-increments  )

        MOV AX, -28 [BP]
        MOV DX, -30 [BP]
        SUB AX, -32 [BP]
        SBB DX, -34 [BP]    ( DX:AX = a^2 - b^2 )

        MOV BX, AX          ( CX:BX = a^2-b^2 )
        MOV CX, DX

        SAR DX, # 0001      ( DX:AX = [a^2 -b^2] /2  )
        RCR AX, # 0001
        ADD AX, BX          ( DX:AX = 3*[a^2-b^2]/2   )
        ADC DX, CX

        SUB AX, -20 [BP]
        SBB DX, -22 [BP]
        SUB AX, -24 [BP]
        SBB DX, -26 [BP]    ( DX:AX = 3*[a^2-b^2]/2 - [dx+dy] )

        SAR DX, # 0001      ( DX:AX = [3*[a^2-b^2]/2-[dx+dy]]/2 )
        RCR AX, # 0001

        ADD -16 [BP],  AX
        ADC -18 [BP], DX    ( update d)

\ loop until y < 0

        POP CX              ( CH,CL = y- annd x-increments )
        POP BX              ( BX =y  )

  HEX
        BEGIN

        CALL SET4PIXELS

        MOV CX, # 0100      ( CH = 1  y-increment  )
                            ( CL = 0  x-increment  )
  DECIMAL
        CMP -18 [BP], # 0000

        0< IF               ( jump if d >= 0 )

        MOV CL, # 1         ( increment in x direction  )

        MOV AX, -20 [BP]
        MOV DX, -22 [BP]
        ADD AX, -40 [BP]
        ADC DX, -42 [BP]    ( DX:AX = dx + 2*b^2 )
        MOV -20 [BP], AX
        MOV -22 [BP], DX    ( dx += 2*b^2  )

        ADD -16 [BP], AX
        ADC -18 [BP], DX    ( d += dx )

        THEN
```



```
        MOV AX, -24 [BP]
        MOV DX, -26 [BP]
        SUB AX, -36 [BP]
        SBB DX, -38 [BP]    ( DX:AX = dy - 2*a^2  )
        MOV -24 [BP], AX
        MOV -26 [BP], DX    ( dy -= 2*a^2  )

        SUB AX, -28 [BP]
        SBB DX, -30 [BP]    ( DX:AX = dy - a^2 )
        SUB -16 [BP], AX
        SBB -18 [BP], DX    ( d += a^2 - dy )

        DEC BX              ( decrement y    )

        0<  UNTIL           ( loop if y >= 0  )

\  restore default Graphics Controller registers
 HEX
        MOV AX, # 0FF08     ( default Bit Mask )
        MOV DX, # 03CE
        OUT DX, AX

        MOV AX, # 0003      ( default Function Select )
        OUT DX, AX

        MOV AX, # 0001      ( default Enable Set/Reset )
        OUT DX, AX

        POP DI
        POP SI
        MOV SP, BP
        POP BP
 DECIMAL
        ADD SP, # 10
        POPIP
 NEXT   END-CODE

HEX

F/H
2VARIABLE FONZ
VARIABLE CHRHGT
A000 CONSTANT IBM

H/F
: NORMAL_VIDEO ( -- )
        PIXEL_MODE 0 SWAP ! ;

: XOR_VIDEO ( -- )
        PIXEL_MODE 18 SWAP ! ;

: OR_VIDEO ( -- )
        PIXEL_MODE 10 SWAP ! ;

: AND_VIDEO ( -- )
        PIXEL_MODE 8 SWAP ! ;

: SET_PIXEL ( -- )
        DIRECT_VIDEO @
        IF      PUT_PIXEGA
        ELSE    PUT_PIXEL
        THEN ;

: READ_PIXEL ( -- )
        DIRECT_VIDEO @
        IF      GET_PIXEGA
```

```
        ELSE    GET_PIXEL
        THEN ;

: 8x14FONT ( -- )
        0200 FONTAD  FONZ 2!  0E CHRHGT ! ;

: 8x16FONT
        0600 FONTAD  FONZ 2!  010 CHRHGT ! ;

: 8x8FONT
        0F000 FA6E   FONZ 2!  08 CHRHGT ! ;

: WPLANE
        2 3C4 PC!  3C5 PC! ;

DECIMAL

: EGA_HI        16 SET_GRAPHMODE 8x14FONT ;
: EGA_LO        14 SET_GRAPHMODE 8x14FONT ;
: VGA_HI        18 SET_GRAPHMODE 8x16FONT ;
: CGA_LO        4 SET_GRAPHMODE ;
: CGA_HI        6 SET_GRAPHMODE ;
: COLO          2 SET_GRAPHMODE ;

TOMODE ON

: CHRPLOT
        F( CHARN  COLOR  X Y  LIML  LIMH  HV /  TBADR #  DX PFLG )
                \ X is 0 to 639, Y is 0 to 349, COLOR
                \ LIML and LIMH are top and bot of window clip limits
                \ DIR : 0,2 -VERTICAL  1,3 -HORIZONTAL
        HV 1 AND
        IF      X Y TO X TO Y
        THEN
        HV DUP 0 = SWAP 3 = OR
        IF      1 TO PFLG
        ELSE    0 TO PFLG
        THEN
        FONZ CHARN  CHRHGT * + TO TBADR

        CHRHGT 0
        DO      X I  PFLG
                IF      NEGATE
                THEN    + DUP DUP TO DX
                LIML < SWAP  LIMH > OR
                TBADR I +  C@L  DUP 0= ROT OR
                IF      DROP
                ELSE  128 8 0
                        DO      2DUP AND
                                IF      DX Y I
                                        CASE    HV 0 = IF  +
                                        ELSE    HV 2 = IF  -
                                        ELSE    HV 3 = IF NEGATE + SWAP
                                        ELSE    + SWAP
                                        ENDCASE
                                 color  PUT_PIXEGA
                                THEN    1 SLR
                        LOOP    2DROP
                THEN
        LOOP ;

 : CHRPLOT_BIOS
        F( CHARN  COLOR  X Y  LIML  LIMH  HV /  TBADR #  DX PFLG )
                \ X is 0 to 639, Y is 0 to 349, COLOR
                \ LIML and LIMH are top and bot of window clip limits
                \ DIR : 0,2 -VERTICAL  1,3 -HORIZONTAL
```



```
        HV 1 AND
        IF      X Y TO X TO Y
        THEN
        HV DUP 0 = SWAP 3 = OR
        IF      1 TO PFLG
        ELSE    0 TO PFLG
        THEN
        FONZ CHARN  CHRHGT * + TO TBADR

        CHRHGT 0
        DO      X I  PFLG
                IF      NEGATE
                THEN    + DUP DUP TO DX
                LIML < SWAP  LIMH > OR
                TBADR I +  C@L  DUP 0= ROT OR
                IF      DROP
                ELSE    128  8 0
                        DO      2DUP AND
                                IF      DX Y I
                                        CASE    HV 0 = IF  +
                                        ELSE    HV 2 = IF  -
                                        ELSE    HV 3 = IF NEGATE + SWAP
                                        ELSE    + SWAP
                                        ENDCASE
                                        color PUT_PIXEL
                                THEN    1 SLR
                        LOOP    2DROP
                THEN
        LOOP ;
: OUTTEXTXY
        F( PTR # CNT COLOR X Y LIML LIMH FG )
                \ Input - string pointer and length
        CNT 0
        DO      PTR I +  C@L
                COLOR X I 8 *
                CASE    FG 3 = IF NEGATE + Y
                ELSE    FG 0 = IF Y SWAP +
                ELSE    FG 2 = IF Y SWAP -
                ELSE    + Y
                ENDCASE
                LIML LIMH FG CHRPLOT
        LOOP  ;

: OUTTEXTXY_BIOS
        F( PTR # CNT COLOR X Y LIML LIMH FG )
                \ Input - string pointer and length
        CNT 0
         DO      PTR I +  C@L
                COLOR X I 8 *
                CASE    FG 3 = IF NEGATE + Y
                ELSE    FG 0 = IF Y SWAP +
                ELSE    FG 2 = IF Y SWAP -
                ELSE    + Y
                ENDCASE
                LIML LIMH FG CHRPLOT_BIOS
        LOOP  ;
: SGPLOTB
        F( PTR #  CNT  COLOR  X  Y  LIM  FG  /  TBADR #  XC  YC )
                \ X is 0 to 79, Y is 0 to 349
                \ FG:  0 bit is vert/horiz;  1 bit is STORE or OR
        FG  2 AND
        IF      [ HEX ] 3 3CE PC!  10 3CF PC! [ DECIMAL ]
        THEN    CNT 0
        DO      PTR I + C@L
                FONZ ROT CHRHGT * + TO TBADR
                COLOR WPLANE
```

```
              FG 1 AND
              IF      X I + Y
              ELSE    X Y I CHRHGT * +
              THEN    466 MIN  TO YC  0 MAX  79 MIN  TO XC
              14 0
              DO      TBADR  I +  C@L
                      YC I + DUP  LIN DUP 0<
                      IF      NEGATE >
                      ELSE    <
                      THEN  0=
                      IF      I8M 80 * XC + 2DUP C@L DROP C!L
                      ELSE    2DROP
                      THEN
              LOOP
       LOOP  [ HEX ]  3 3CE PC!  0 3CF PC!  [ DECIMAL ] ;

: LINE F( X1 Y1 X2 Y2 COLOR )
       CASE    X1 X2 =                    \ VERTICAL LINE x1 = x2
               IF      X1 Y1 Y2 COLOR VERT_LINE
       ELSE    Y1 Y2 =                    \ TEST FOR HORIZ LINE y1 = y2
               IF      Y1 X1 X2 COLOR HORIZ_LINE
       ELSE    Y1 Y2 - ABS X1 X2 - ABS            \ dy , dx
               > IF    X1 Y1 X2 Y2 COLOR HS_LINE  \ dx > dy ->  SLOPE > 1
               ELSE    X1 Y1 X2 Y2 COLOR LS_LINE  \ dx <= dy -> SLOPE <= 1
               THEN
       ENDCASE ;

 TOMODE ON

: LINE_BIOS F( x1 y1 x2 y2 color# / m1 m2 )
      x2 x1 - to m1 y2 y1 - to m2

     m1 0= if  m2 0<  if  y2 y1 do i  x1 swap color# put_pixel -1
                          +loop

                      else  m2 0= if x1 x1 y1 y1 color# put_pixel
                                 else    y2 y1 do i x1 swap color# put_pixel
                                         loop
                                 then
                            then
           else  x2 x1 > if m1 0 do i DUP  x1 + swap m2 m1 */ y1 +
                                      color# put_pixel
                                     loop
                         else m2 0< if m1 0 do i DUP  x1 + swap m2 m1 */ y1 +
                                                color# put_pixel -1
                                               +loop
                                    else m1 0 do i DUP x1 + swap m2 m1 */ abs
                                                 y1 +
                                                 color# put_pixel  -1
                                               +loop
                                    then
                         then
         THEN ;

: VERT_LINE_BIOS  ( x y1 y2 color# -- )
   F( x y1 y2 color#  )

   X Y1 X Y2 COLOR# LINE_BIOS ;

 : HORIZ_LINE_BIOS  ( y x1 x2 color# -- )
   F( Y X1 X2 COLOR# )
   X1 Y X2 Y COLOR# LINE_BIOS ;

TYPE> AXISPARAM
      INTEGER   S_LEN  L_LEN  A_COLOR  N_COLOR  N_SPACE
ENDTYPE> AXISPARAM
```



```
AXISPARAM   HAXIS  VAXIS
2 DUP TO HAXIS S_LEN    TO VAXIS S_LEN
4 DUP TO HAXIS L_LEN    TO VAXIS L_LEN
7 DUP TO HAXIS A_COLOR  TO VAXIS A_COLOR
7 DUP TO HAXIS N_COLOR  TO VAXIS N_COLOR
5 3 TO HAXIS N_SPACE    TO VAXIS N_SPACE

TYPE> SCALERPARAM
      INTEGER  SM_TIC  LG_TIC  NUMERIC  PTS
      DOUBLE.REAL  S_OFS  L_OFS  N_OFS
ENDTYPE> SCALERPARAM

SCALERPARAM   SCALER1

TYPE> GRIDPARAM
      INTEGER   G_COLOR  G_LEN
ENDTYPE> GRIDPARAM

GRIDPARAM   HGRID VGRID
100 DUP   TO HGRID G_LEN     TO VGRID G_LEN
7 DUP     TO HGRID G_COLOR   TO VGRID G_COLOR

VARIABLE SHOWGRID
SHOWGRID OFF

: INT ( dr -- dr' )
       NIP 0 SWAP ;
: TRUNC ( dr -- n )
       NIP ;
: ROUND ( dr -- n )
       SWAP 0< - ;

: FIXPTERR
       ." FIXED POINT ARITHMETIC ERROR IN "  R>
       BEGIN   2- DUP >NAME
               [ -1 >NAME ] LITERAL <>
       UNTIL  >NAME  DUP YC@  ?LINE  .ID QUIT ;

: 10^N ( n -- 10^n )
       >R 10 1  R@ ABS DUP 5 <
       IF      0 ?DO   OVER *
               LOOP    NIP  R> 0<
               IF      256 256 ROT */ 0
               ELSE    0 SWAP
               THEN
       ELSE    2DROP FIXPTERR
       THEN  ;

: *10^N ( dr n -- dr )
       10^N DR* ;

: /10^N ( dr n -- dr )
       NEGATE *10^N ;

: CALTIC_OFS                    \ INPUT (X1,SM_OFS TIC_LOCS,TIC )
       F( ST_VAL # SM_OFS # TIC_LOC TICK / CT_TEMP )
       ST_VAL SM_OFS D+ TRUNC TO TEMP
       0 TO CT                          \ OUTPUT (TIC_OFS)
       BEGIN   TEMP TIC_LOC MOD 0<>
       WHILE   CT INCR
               TEMP TICK + TO TEMP
       REPEAT
       SM_OFS 0 CT TICK * D+ ;

: CALPOWER
```

```
        F( DX # THOLD # / EXP )         \ INPUT = dx :  OUTPUT = exponent
        0 TO EXP
        DX THOLD
        D< IF   BEGIN   EXP DECR
                        DX  THOLD EXP *10^N D> EXP -4 < OR
                UNTIL
        ELSE    BEGIN   EXP INCR
                        DX  THOLD EXP *10^N D< EXP 3 > OR
                UNTIL
                EXP DECR
        THEN
        EXP -4 <  EXP 3 > OR
        IF      COBO CR FASTTYPE EXP .
                TRUE THROW \ ABORT" ERR1 - AUTOSCALER OUT OF RANGE " \ AB2
        THEN
        EXP ;

: AUTO_RANGE
        F( DX # THOLD # / EXP )     \ INPUT : dx, 10^N OF dx
        DX THOLD CALPOWER DUP TO EXP    \ OUTPUT: SM_TIC, LG_TIC, NUMERIC
        CASE    DX 2.0 THOLD DR* EXP *10^N D<
                IF      \ RANGE 1 : dx < 60.0 * 10^N
                        1    \ SM_TIC INTERVAL
                        5    \ LG_TIC INTERVAL
                        10    \ NUMERIC INTERVAL
        ELSE    DX  4.0 THOLD DR* EXP *10^N D<
                IF      \ RANGE 2 : dx < 120.0 * 10^N
                        2  10 20
        ELSE    5 10 50                 \ RANGE 3 : dx < 300.0 * 10^N
        ENDCASE ;

: START_LOC
        F( P # OFS # LOC )
        P OFS DR* 0 LOC D+ ;

: INCR_TIC
        F( I TIC N DXP # T_START # )
        0 I TIC * DXP DR/ 0 N DR* T_START D+ ROUND ;

: TIC_PARAM
        F( DX # EXP LOC L1 # P ' SCALERPARAM / DXP # )
        DX EXP /10^N 2DUP                   \  DXP
        0 P PTS 2SWAP DR/ TO DXP
        DXP P S_OFS LOC  START_LOC          \  S_START
        DXP P L_OFS LOC  START_LOC          \  L_START
        DXP P N_OFS LOC  START_LOC          \  N_START
        L1 EXP /10^N P N_OFS D+ EXP *10^N ;  \  N_VAL

: DRAWGRID
        F( G_START # LG_TIC DXP # N DN L1 HV GP ' GRIDPARAM
          / LASTPT LEN KOLOR )
        N DN + TO LASTPT
        GP G_LEN TO LEN
        GP G_COLOR TO KOLOR
        DXP 0 LG_TIC DR/ TRUNC 1+ 0          \ NUMBER OF GRID LINES TO PLOT
        DO      T LG_TIC DN DXP G_START  INCR_TIC DUP DUP
                LASTPT <  SWAP N > AND
                IF      L1 DUP LEN HV
                        IF      - KOLOR VERT_LINE   \ HV - GIRD ORIENTATION
                        ELSE    + KOLOR HORIZ_LINE  \ 1 : VERTICAL
                        THEN                        \ 0 : HORIZONTAL
                ELSE    DROP
                THEN
        LOOP ;

: DRAWGRID_BIOS
```



```
        F( G_START # LG_TIC DXP # N DN L1 HV GP ' GRIDPARAM
          / LASTPT LEN KOLOR )
        N DN + TO LASTPT
        GP G_LEN TO LEN
        GP G_COLOR TO KOLOR
        DXP 0 LG_TIC DR/ TRUNC 1+ 0          \ NUMBER OF GRID LINES TO PLOT
        DO      T LG_TIC DN DXP G_START  INCR_TIC DUP DUP
                LASTPT <  SWAP N > AND
                IF      L1 DUP LEN HV
                        IF      - KOLOR VERT_LINE_BIOS   \ HV - GIRD ORIENTATION
                        ELSE    + KOLOR HORIZ_LINE_BIOS  \ 1 : VERTICAL
                        THEN                             \ 0 : HORIZONTAL
                ELSE    DROP
                THEN
        LOOP ;


: PLOTTIC
        F( DXP # S_START # L_START # SM_TIC LG_TIC PTS LASTPT X Y HV
             AP ' AXISPARAM / T_LOC L_LOC K KOLOR SL LL )
        AP A_COLOR TO KOLOR
        AP S_LEN TO SL
        AP L_LEN TO LL   HV
        IF      Y X LASTPT KOLOR  HORIZ_LINE
                X Y TO X TO Y     ( SWAP X AND Y FOR VERTICAL TIC MARKS )
        ELSE    X Y LASTPT KOLOR  VERT_LINE
        THEN
        1 TO K
        L_START ROUND TO L_LOC
        DXP 0 SM_TIC DR/ TRUNC 1+ 0
        DO      T SM_TIC PTS DXP S_START INCR_TIC DUP TO T_LOC
                L_LOC =
                IF      T_LOC DUP Y > SWAP LASTPT <  AND
                        IF      T_LOC X DUP   HV
                                IF      LL - KOLOR VERT_LINE  \ PLOT LARGE TIC
                                ELSE    LL + KOLOR  HORIZ_LINE
                                THEN
                        THEN
                        K LG_TIC PTS DXP L_START INCR_TIC TO L_LOC
                        K INCR
                ELSE    T_LOC DUP Y > SWAP LASTPT < AND
                        IF      T_LOC X DUP   HV
                                IF      SL - KOLOR VERT_LINE  \ PLOT SMALL TIC
                                ELSE    SL + KOLOR  HORIZ_LINE
                                THEN
                        THEN
                THEN
        LOOP  ;

: PLOTTIC_BIOS
        F( DXP # S_START # L_START # SM_TIC LG_TIC PTS LASTPT X Y HV
             AP ' AXISPARAM / T_LOC L_LOC K KOLOR SL LL )
        AP A_COLOR TO KOLOR
        AP S_LEN TO SL
        AP L_LEN TO LL   HV
        IF      Y X LASTPT KOLOR  HORIZ_LINE_BIOS
                X Y TO X TO Y     ( SWAP X AND Y FOR VERTICAL TIC MARKS )
        ELSE    X Y LASTPT KOLOR  VERT_LINE_BIOS
        THEN
        1 TO K
        L_START ROUND TO L_LOC
        DXP 0 SM_TIC DR/ TRUNC 1+ 0
        DO      T SM_TIC PTS DXP S_START INCR_TIC DUP TO T_LOC
                L_LOC =
                IF      T_LOC DUP Y > SWAP LASTPT <  AND
                        IF      T_LOC X DUP   HV
```

```
                        IF      LL - KOLOR VERT_LINE_BIOS  \ PLOT LARGE T
IC
                        ELSE    LL + KOLOR  HORIZ_LINE_BIOS
                        THEN
                    THEN
                    K LG_TIC PTS DXP L_START INCR_TIC TO L_LOC
                    K INCR
                ELSE    T_LOC DUP Y > SWAP LASTPT < AND
                        IF      T_LOC X DUP   HV
                                IF      SL - KOLOR VERT_LINE_BIOS  \ PLOT SMALL T
IC
                                ELSE    SL + KOLOR  HORIZ_LINE_BIOS
                                THEN
                        THEN
                THEN
        LOOP  ;


: DR.$ ( dr exp -- string )
        0 SWAP NEGATE 1- 0 MAX DR>$
        3DUP DUP 1- CA$ ASCII . =
        IF      1-
        THEN ;

: STRLOC
        8 2 */ ;

: PLOTLABELS
        F( DXP # EXP NUMERIC N_START # N_VAL # N FIRSTPT LASTPT M HV
                P ' AXISPARAM 7 LOC STRNUM )
        DXP 0 NUMERIC DR/ TRUNC 1+ 0
        DO      I NUMERIC N DXP N_START INCR_TIC DUP DUP TO LOC
                STRNUM STRLOC - FIRSTPT > SWAP STRNUM STRLOC + LASTPT < AND
                IF      N_VAL 0 NUMERIC I * EXP *10 N D+
                        EXP DR.$ DUP TO STRNUM P N_COLOR
                        LOC STRNUM STRLOC - M P N_SPACE  HV
                        IF      + 0 349
                        ELSE    - SWAP 0 639
                        THEN
                        HV OUTTEXTXY
                THEN
        LOOP ;

 : PLOTLABELS_BIOS
        F( DXP # EXP NUMERIC N_START # N_VAL # N FIRSTPT LASTPT M HV
                P ' AXISPARAM 7 LOC STRNUM )
        DXP 0 NUMERIC DR/ TRUNC 1+ 0
        DO      I NUMERIC N DXP N_START INCR_TIC DUP DUP TO LOC
                STRNUM STRLOC - FIRSTPT > SWAP STRNUM STRLOC + LASTPT < AND
                IF      N_VAL 0 NUMERIC I * EXP *10 N D+
                        EXP DR.$ DUP TO STRNUM P N_COLOR
                        LOC STRNUM STRLOC - M P N_SPACE  HV
                        IF      + 0 349
                        ELSE    - SWAP 0 639
                        THEN
                        HV OUTTEXTXY_BIOS
                THEN
        LOOP ;


: HORIZ_AXIS
        F( X1 # DX # EXP P ' SCALERPARAM  X Y
                / S_START # L_START # N_START # DXP # N_VAL # LASTPT )
  \ ************  CALCULATE PLOTTING PARAMS **********
        DX EXP X X1 P '    TIC_PARAM
        TO N_VAL     TO N_START    TO L_START    TO S_START  TO DXP
```



```
        X P PTS + TO LASTPT
  \ ***************  PLOT TICKS  ********************
        DXP S_START L_START P SM_TIC P LG_TIC P PTS LASTPT X Y 1 HAXIS '
        PLOTTIC
  \ ***************  PLOT LABELS  *******************
        DXP EXP P NUMERIC N_START N_VAL P PTS X LASTPT Y 1 HAXIS '
        PLOTLABELS
  \ ***************  PLOT GRID  *********************
        SHOWGRID
        IF      L_START P LG_TIC DXP X P PTS Y 1 HGRID '
                DRAWGRID
        THEN ;

: HORIZ_AXIS_BIOS
        F( X1 # DX # EXP P ' SCALERPARAM  X Y
                / S_START # L_START # N_START # DXP # N_VAL # LASTPT )
  \ ************  CALCULATE PLOTTING PARAMS **********
        DX EXP X X1 P '    TIC_PARAM
        TO N_VAL     TO N_START    TO L_START    TO S_START  TO DXP
        X P PTS + TO LASTPT
  \ ***************  PLOT TICKS  ********************
        DXP S_START L_START P SM_TIC P LG_TIC P PTS LASTPT X Y 1 HAXIS '
        PLOTTIC_BIOS
  \ ***************  PLOT LABELS  *******************
        DXP EXP P NUMERIC N_START N_VAL P PTS X LASTPT Y 1 HAXIS '
        PLOTLABELS_BIOS
  \ ***************  PLOT GRID  *********************
        SHOWGRID
        IF      L_START P LG_TIC DXP X P PTS Y 1 HGRID '
                DRAWGRID_BIOS
        THEN ;


: VERT_AXIS
        F( X1 # DX # EXP P ' SCALERPARAM X Y
                / S_START # L_START # N_START # DXP # N_VAL # LASTPT )
  \ ************  CALCULATE PLOTTING PARAMS **********
        DX EXP Y X1 P '  TIC_PARAM
        TO N_VAL     TO N_START    TO L_START    TO S_START  TO DXP
        Y P PTS + TO LASTPT
  \ ***************  PLOT TICKS  ********************
        DXP S_START L_START P SM_TIC P LG_TIC P PTS LASTPT X Y 0 VAXIS '
        PLOTTIC
  \ ***************  PLOT LABELS  *******************
        DXP EXP P NUMERIC N_START N_VAL P PTS Y LASTPT X 0 VAXIS '
        PLOTLABELS
  \ ***************  PLOT GRID  *********************
        SHOWGRID
        IF      L_START P LG_TIC DXP Y P PTS X 0 VGRID '
                DRAWGRID
        THEN ;

 : VERT_AXIS_BIOS
        F( X1 # DX # EXP P ' SCALERPARAM X Y
                / S_START # L_START # N_START # DXP # N_VAL # LASTPT )
  \ ************  CALCULATE PLOTTING PARAMS **********
        DX EXP Y X1 P '  TIC_PARAM
        TO N_VAL     TO N_START    TO L_START    TO S_START  TO DXP
        Y P PTS + TO LASTPT
  \ ***************  PLOT TICKS  ********************
        DXP S_START L_START P SM_TIC P LG_TIC P PTS LASTPT X Y 0 VAXIS '
        PLOTTIC_BIOS
  \ ***************  PLOT LABELS  *******************
        DXP EXP P NUMERIC N_START N_VAL P PTS Y LASTPT X 0 VAXIS '
        PLOTLABELS_BIOS
  \ ***************  PLOT GRID  *********************
```

```
        SHOWGRID
        IF      L_START P LG_TIC DXP Y P PTS X 0 VGRID
                DRAWGRID_BIOS
        THEN ;

: AUTOSCALE
        F( X1 # X2 # X Y COLOR NPT HV / DX # EXP S_VAL # )
        X2 X1 D- 2DUP TO DX
        60.0 NPT 480 */                 \ THRESHOLD VALUE
        AUTO_RANGE TO SCALER1 NUMERIC   \ AUTORANGE INPUT - DX, THRESHOLD
        TO SCALER1 LG_TIC
        TO SCALER1 SM_TIC
        TO EXP
        X1 EXP /10^N TO S_VAL  \ S_VAL = x1 / P_NTH
                               \ S_OFS = (INT(S_VAL/SM_TIC)+1)*SM_TIC - S_VAL
        S_VAL 0 SCALER1 SM_TIC DR/ INT 1.0 D+ 0 SCALER1 SM_TIC DR* S_VAL D-
        TO SCALER1 S_OFS
        S_VAL SCALER1 S_OFS SCALER1 LG_TIC SCALER1 SM_TIC CALTIC_OFS
        TO SCALER1 L_OFS
        S_VAL SCALER1 S_OFS SCALER1 NUMERIC SCALER1 SM_TIC CALTIC_OFS
        TO SCALER1 N_OFS
        8x8FONT
        NPT TO SCALER1 PTS
        X1 DX EXP SCALER1 ^ X Y                 \ HV - HORIZ/VERT POSITION
    HV IF   COLOR TO HAXIS A_COLOR  HORIZ_AXIS \       FLAG ( 1 : HORIZ
       ELSE COLOR TO VAXIS A_COLOR  VERT_AXIS  \            ( 0 : VERT
       THEN
     ;

: AUTOSCALE_BIOS
        F( X1 # X2 # X Y COLOR NPT HV / DX # EXP S_VAL # )
        X2 X1 D- 2DUP TO DX
        60.0 NPT 480 */                 \ THRESHOLD VALUE
        AUTO_RANGE TO SCALER1 NUMERIC   \ AUTORANGE INPUT - DX, THRESHOLD
        TO SCALER1 LG_TIC
        TO SCALER1 SM_TIC
        TO EXP
        X1 EXP /10^N TO S_VAL  \ S_VAL = x1 / P_NTH
                               \ S_OFS = (INT(S_VAL/SM_TIC)+1)*SM_TIC - S_VAL
        S_VAL 0 SCALER1 SM_TIC DR/ INT 1.0 D+ 0 SCALER1 SM_TIC DR* S_VAL D-
        TO SCALER1 S_OFS
        S_VAL SCALER1 S_OFS SCALER1 LG_TIC SCALER1 SM_TIC CALTIC_OFS
        TO SCALER1 L_OFS
        S_VAL SCALER1 S_OFS SCALER1 NUMERIC SCALER1 SM_TIC CALTIC_OFS
        TO SCALER1 N_OFS
        8x8FONT
        NPT TO SCALER1 PTS
        X1 DX EXP SCALER1 ^ X Y                 \ HV - HORIZ/VERT POSITION
    HV IF   COLOR TO HAXIS A_COLOR  HORIZ_AXIS_BIOS \      FLAG ( 1 : HORIZ
       ELSE COLOR TO VAXIS A_COLOR  VERT_AXIS_BIOS  \           ( 0 : VERT
       THEN
     ;

\ TEST AND EXAMPLE WORDS

: AST
        F( X1 # X2 # )
        VGA HI
        SHOWGRID ON
        80 TO HGRID G_LEN
        X1 X2 0 100 12 512 1 AUTOSCALE
        X1 X2 0 200 13 256 1 AUTOSCALE
        X1 X2 0 300 14 128 1 AUTOSCALE
        80 TO VGRID G_LEN
        X1 X2 530 20 -15 320 0 AUTOSCALE
        X1 X2 400 160 11 160 0 AUTOSCALE
        ;

: AST_BIOS
        F( X1 # X2 # )
        VGA HI
        SHOWGRID ON
        80 TO HGRID G_LEN
        X1 X2 0 100 12 512 1 AUTOSCALE_BIOS
        X1 X2 0 200 13 256 1 AUTOSCALE_BIOS
        X1 X2 0 300 14 128 1 AUTOSCALE_BIOS
        80 TO VGRID G_LEN
        X1 X2 530 20 -15 320 0 AUTOSCALE_BIOS
        X1 X2 400 160 11 160 0 AUTOSCALE_BIOS
        ;
: EDF
        CO80 ED ;

: TXT
        VGA HI
        $" VERTICAL 1" 11 300 200 0 639 0 OUTTEXTXY
        $" VERTICAL 2" 12 300 200 0 639 2 OUTTEXTXY
        $" HORIZONTAL 1" 13 300 200 0 349 1 OUTTEXTXY
        $" HORIZONTAL 2" 14 300 200 0 349 3 OUTTEXTXY ;

 : TXT_BIOS
        VGA HI
        $" VERTICAL 1" 11 300 200 0 639 0 OUTTEXTXY_BIOS
        $" VERTICAL 2" 12 300 200 0 639 2 OUTTEXTXY_BIOS
        $" HORIZONTAL 1" 13 300 200 0 349 1 OUTTEXTXY_BIOS
        $" HORIZONTAL 2" 14 300 200 0 349 3 OUTTEXTXY_BIOS ;

INTEGER A1
: AMERICA F( X Y )
        $" UNITED STATES OF AMERICA" 11 X Y 0 639 0 OUTTEXTXY
        $" UNITED STATES OF AMERICA" 12 X Y 0 639 2 OUTTEXTXY
        $" UNITED STATES OF AMERICA" 13 X Y 0 349 1 OUTTEXTXY
        $" UNITED STATES OF AMERICA" 14 X Y 0 349 3 OUTTEXTXY ;
          INTEGER A1
: AMERICA_BIOS F( X Y )
        $" UNITED STATES OF AMERICA" 11 X Y 0 639 0 OUTTEXTXY_BIOS
        $" UNITED STATES OF AMERICA" 12 X Y 0 639 2 OUTTEXTXY_BIOS
        $" UNITED STATES OF AMERICA" 13 X Y 0 349 1 OUTTEXTXY_BIOS
        $" UNITED STATES OF AMERICA" 14 X Y 0 349 3 OUTTEXTXY_BIOS ;


 \  UNIFORM - Produces 16 bit uniformly distributed r.n.'s

: UNIFORM       ( n -- n n' )
        DUP 12345 *  4567 + ;

VARIABLE RNSTATE   1234 RNSTATE !

: URANS         ( AD N -- )
        RNSTATE @  -ROT  0
        DO      >R                      \ rn      | ad
                UNIFORM                 \ rn rn'
                SWAP R> !++
        LOOP  DROP  RNSTATE ! ;

: RANDOMVECT    ( max k -- r1 .. rk )
        RNSTATE @  SWAP 0               \ max rn k 0
        DO      UNIFORM >R              \ max rn        | rn'
                OVER UM* NIP SWAP R>    \ rout max rn'
        LOOP    RNSTATE ! DROP ;

\ SAMPLE DEMO GRAPHICS
```

```
 : TLINE_BIOS  F( XX YY. )
\  XX= MAXIMUM AXIS LENGTH OF LINE_BIOS TESTBOX
\  YY= LINE_BIOS TESTBOX'S UPPER LEFT CORNER (X,Y=X) COORDINATE
        BEGIN   XX 2 RANDOMVECT YY + SWAP YY + XX 2 RANDOMVECT YY + SWAP YY +
                15 1 RANDOMVECT   LINE_BIOS  KEY?
        UNTIL   ?KEYPAUSE ;

: TEST2  F( X1 Y1 XX )
BEGIN  X1 Y1 XX 2 RANDOMVECT Y1 - ABS SWAP X1 - ABS SWAP 15 1 RANDOMVECT LINE
 X1 Y1 XX  2 RANDOMVECT Y1 - ABS SWAP X1 + SWAP  15 1 RANDOMVECT  LINE
 X1 Y1 XX  2 RANDOMVECT Y1 + SWAP X1 + SWAP  15 1 RANDOMVECT LINE
 X1 Y1 XX  2 RANDOMVECT Y1 +  SWAP X1 - ABS SWAP  15 1 RANDOMVECT LINE KEY?
  UNTIL   ?KEYPAUSE ;
VARIABLE A1 VARIABLE B1 VARIABLE AB VARIABLE BB VARIABLE CC VARIABLE DD
TOMODE ON

 : FIREWORK1
  BEGIN
  VGA HI
 11 SET_BORDER
  40 0 DO
  600 1 RANDOMVECT TO A1  400 1 RANDOMVECT TO B1  130 1 RANDOMVECT TO AB

  20 0 DO  A1 B1 AB  2 RANDOMVECT B1 - ABS SWAP A1 - ABS SWAP 15 1 RANDOMVECT
  LINE  A1 B1 AB  2 RANDOMVECT B1 - ABS SWAP A1 + SWAP  15 1 RANDOMVECT LINE
 A1 B1 AB  2 RANDOMVECT B1 + SWAP A1 + SWAP  15 1 RANDOMVECT  LINE
  A1 B1 AB  2 RANDOMVECT B1 +  SWAP A1 - ABS SWAP 15 1 RANDOMVECT  LINE
   LOOP  LOOP KEY?  UNTIL  ?KEYPAUSE ;

 : FIREWORK2

  BEGIN  VGA HI  14 SET_BORDER 40 0 DO
I TO BB 600 1 RANDOMVECT TO A1 400 1 RANDOMVECT TO B1 130 1 RANDOMVECT TO AB

  20 0 DO  A1 B1 AB  2 RANDOMVECT B1 - ABS SWAP A1 - ABS SWAP BB LINE_BIOS
         A1 B1 AB  2 RANDOMVECT B1 - ABS SWAP A1 + SWAP  BB   LINE_BIOS

         A1 B1 AB  2 RANDOMVECT B1 + SWAP A1 + SWAP  BB   LINE_BIOS

         A1 B1 AB  2 RANDOMVECT B1 +  SWAP A1 - ABS SWAP  BB   LINE_BIOS

         LOOP   LOOP KEY? UNTIL  ?KEYPAUSE ;
: FIREWORK3
  BEGIN
  VGA HI
  12 SET_BORDER
  40 0 DO
  600 1 RANDOMVECT TO A1  400 1 RANDOMVECT TO B1  130 1 RANDOMVECT TO AB
   15 1 RANDOMVECT TO BB    15 1 RANDOMVECT TO CC   15 1 RANDOMVECT TO DD
  20 0 DO  A1 B1 AB  2 RANDOMVECT B1 - ABS SWAP A1 - ABS SWAP BB  LINE
        A1 B1 AB  2 RANDOMVECT B1 - ABS SWAP A1 + SWAP  BB   LINE
 A1 B1 AB  2 RANDOMVECT B1 + SWAP A1 + SWAP  BB   LINE
A1 B1 AB  2 RANDOMVECT B1 +  SWAP A1 - ABS SWAP  BB  LINE LOOP
   30 0 DO  A1 B1 AB 2/  2 RANDOMVECT B1 - ABS SWAP A1 - ABS SWAP CC   LINE
         A1 B1 AB 2/  2 RANDOMVECT B1 - ABS SWAP A1 + SWAP  CC   LINE
 A1 B1 AB 2/  2 RANDOMVECT B1 + SWAP A1 + SWAP  CC   LINE
A1 B1 AB 2/  2 RANDOMVECT B1 +  SWAP A1 - ABS SWAP  CC   LINE LOOP
 30 0 DO  A1 B1 AB 2/ 2/  2 RANDOMVECT B1 - ABS SWAP A1 - ABS SWAP DD   LINE
        A1 B1 AB 2/ 2/ 2 RANDOMVECT B1 - ABS SWAP A1 + SWAP  DD   LINE
        A1 B1 AB 2/ 2/  2 RANDOMVECT B1 + SWAP A1 + SWAP  DD  LINE
        A1 B1 AB 2/ 2/  2 RANDOMVECT B1 +  SWAP A1 - ABS SWAP  DD   LINE
LOOP   LOOP KEY? UNTIL ?KEYPAUSE ;

  : TLINE  F( XX YY )
BEGIN   XX 5 RANDOMVECT
```



```
        4 ROLL YY + 4 ROLL YY + 4 ROLL YY + 4 ROLL YY +   4 ROLL  LINE  KEY?
       UNTIL   ?KEYPAUSE ;

: TELLIPSEA 480 1 RANDOMVECT TO A1  600 1 RANDOMVECT TO B1
            50 1 RANDOMVECT 1 + TO AB  AB 0 DO I 2 AND SET_ACTIVEPAGE
            I 15 AND I DUP SWAP A1 B1 ELLIPSE LOOP ;


: TELLIPSE   BEGIN
             TELLIPSEA KEY? UNTIL ?KEYPAUSE ;
: TESTA
  40 0 DO
 600 1 RANDOMVECT TO A1  400 1 RANDOMVECT TO B1  130 1 RANDOMVECT TO AB
  15 1 RANDOMVECT TO BB    15 1 RANDOMVECT TO CC   15 1 RANDOMVECT TO DD
 20 0 DO  A1 B1 AB  2 RANDOMVECT B1 - ABS SWAP A1 - ABS SWAP BB  LINE
       A1 B1 AB  2 RANDOMVECT B1 - ABS SWAP A1 + SWAP  BB   LINE
 A1 B1 AB  2 RANDOMVECT B1 + SWAP A1 + SWAP  BB   LINE
 A1 B1 AB  2 RANDOMVECT B1 + SWAP A1 - ABS SWAP  BB  LINE LOOP
  30 0 DO  A1 B1 AB 2/  2 RANDOMVECT B1 - ABS SWAP A1 - ABS SWAP CC   LINE
           A1 B1 AB 2/  2 RANDOMVECT B1 - ABS SWAP A1 + SWAP  CC   LINE
           A1 B1 AB 2/  2 RANDOMVECT B1 + SWAP A1 + SWAP  CC   LINE
A1 B1 AB 2/  2 RANDOMVECT B1 +  SWAP A1 - ABS SWAP  CC   LINE LOOP
 30 0 DO  A1 B1 AB 2/ 2/  2 RANDOMVECT B1 - ABS SWAP A1 - ABS SWAP DD   LINE
       A1 B1 AB 2/ 2/ 2 RANDOMVECT B1 - ABS SWAP A1 + SWAP  DD   LINE
       A1 B1 AB 2/ 2/  2 RANDOMVECT B1 + SWAP A1 + SWAP  DD  LINE
       A1 B1 AB 2/ 2/  2 RANDOMVECT B1 +  SWAP A1 - ABS SWAP  DD   LINE
LOOP   LOOP ;
: TESTB BEGIN 2 0 DO I 1 - SET_ACTIVEPAGE TESTA LOOP KEY? UNTIL ?KEYPAUSE ;
```

DISTRIBUTION

	Copies
Chief of Naval Operations Department of the Navy ATTN: OP-392C Washington, DC 20350	1
COMDESRON-15 Attn: LCD Steve Anthony FPO San Francisco CA 96601-4717	1
COMDESRON-20 Attn: CAPT Kaiser FPO Miami FL 34099-4719	1
Defense Technical Information Center Cameron Station Alexandria, VA 22304-61451	12
Commander Fleet Training Group Western Pacific FPO Seattle WA 98782	2
Internal Distribution	
E231	2
E232	3
U	1
U02	1
U20	1
U25	3
U25 (Craun, P.)	1
U25 (Craun, P. J.)	1
U25 (Davis)	1
U25 (French)	1
U25 (Ko)	7
U25 (Rosenbaum)	2
U25 (Yan)	1

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	14 June 1991	FINAL

4. TITLE AND SUBTITLE
FORTH GRAPHICS TOOLBOX (A USER'S GUIDE FOR USE WITH RFF FORTH)

5. FUNDING NUMBERS

6. AUTHOR(S)
Hanseok Ko

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Surface Warfare Center
10901 New Hampshire Avenue
Silver Spring, MD 20903-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NAVSWC MP 91-404

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

FORTH GRAPHICS TOOLBOX has a rich collection of graphics routines immediately useful for all FORTH based application software running on IBM-PC clone microcomputers. The routines are built upon graphics related primitives of both video BIOS call functions and direct-video functions. The user can develop more exotic application software based on the routines listed in this package.

14. SUBJECT TERMS
Graphics, Routines, Software, Applications, FORTH, Underwater acoustics signal processing

15. NUMBER OF PAGES
65

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and its title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

BLOCK 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If Known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief ***(Maximum 200 words)*** factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code ***(NTIS only)***

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.